Office of Operations Research and Resource Analysis

DORRA

"Insight through Analysis"

Final Report

DEVELOPMENT OF DCMC SITE SELECTION MODEL

MAY 1998

For
Defense Contract Management Command
8726 John J. Kingman Road
Ft. Belvoir, VA 22060-6221

DLA-98-PB80140

Office of Operations Research and Resource Analysis

DORRA

"Insight through Analysis"

Final Report

DEVELOPMENT OF DCMC SITE SELECTION MODEL

MAY 1998

For
Defense Contract Management Command
8726 John J. Kingman Road
Ft. Belvoir, VA 22060-6221

DLA-98-PB80140

DLA-98-PB80140

DEFENSE CONTRACT MANAGEMENT COMMAND
SITE SELECTION MODEL

Major Randy Zimmerman
Captain Jeffery L. Huisingh

MAY 1998

DEPARTMENT OF DEFENSE
DEFENSE LOGISTICS AGENCY
Office of Operations Research and
Resource Analysis

c/o Defense Supply Center Richmond
8000 Jefferson Davis Highway
Richmond, VA 23297-5082

DEFENSE LOGISTICS AGENCY
OPERATIONS RESEARCH AND RESOURCE ANALYSIS
DORRA
c/o DEFENSE SUPPLY CENTER RICHMOND
RICHMOND, VIRGINIA 23297-5082

IN REPLY
REFER TO DORRA

FOREWORD

This report documents a new automated method for selection of meeting and training event locations. The program developed for this project, OffSite, optimizes the selection of training locations by a modified use of Dykstra's algorithm, used to minimize the travel costs between more than 260 cities across the United States. OffSite considers both travel and per diem costs for all potential training locations.

We would like to thank Mr. Steve Herlihy of the Defense Contract Management Command for his guidance and assistance with this project. Additionally, we would like to thank Captain Jeff Huisingh of TRAC - Monterey and Mr. Harold Yamauchi of Rolands & Associates Corporation. Their work on this project was key to the success of this project.

JOHN E. FIRTH
Colonel, USA
Chief, DLA Office of Operations Research
and Resource Analysis

EXECUTIVE SUMMARY

The Defense Contract Management Command (DCMC) Workforce Strategy team asked the DLA Office of Operations Research and Resource Analysis (DORRA) office to develop a methodology that would select the least cost training sites for DCMC employees. The resulting program, Offsite, works by optimizing the airfare, and individually the meals and incidental expense and lodging cost elements of the per diem costs. A conservative estimate of possible model savings is about $400,000 annually if a level of savings of 5% are achieved. Actual savings could be even higher.

All of the data used in this model are available to the public on the Internet. In the past, such data were not readily available in electronic form. Plans are underway to make the model available to all potential DLA users on the DCMC website. This will help ensure the best possible distribution, availability, and usage to further maximize the potential savings.

Every effort was made to make the OffSite program easy to use. An intuitive graphical user interface requiring few user inputs allows a typical user to have results in a minute or less. The drop-down menus and other features of the user screens are of the same grade and quality as the best commercially available software.

OffSite uses a database of 4,640 GSA contracted airfare rates to solve for the 2^{261} possible origin-destination airport pair combinations. OffSite uses a modification of Dykstra's algorithm to select the optimal route from among thousands of possibilities.

The problem of finding the most cost-effective location for a training course, meeting, or conference is not unique to DCMC. It is a problem for the entire Federal government. The alternative to using this model, a manual site selection method, seldom yields the best solution when considering several alternatives. Frequently, the best solution can be counter-intuitive. For example, the optimal site can change, even though the candidate sites remain unchanged, if the number meeting day's changes or the number of people coming from each origin changes. Thus, selection experience with familiar candidate cities does not help the manual selection process.

The potential savings through possible government-wide application are enormous. The Federal government spends about $7 billion annually on travel, about half of which is for DoD. It is possible that $2 billion, or more, of the $7 billion is for the type of training and meetings that have discretionary site locations. If so, Internet availability and promotion of the model through reengineering laboratories could result in government wide savings of tens of millions, or possibly even more. Currently DORRA is exploring reinvention laboratory possibilities that could promote government wide use.

TABLE OF CONTENTS

SECTION 1 - STUDY DESCRIPTION

PROBLEM STATEMENT

Develop a site selection model to assist the DCMC management in selecting cost-effective training locations.

BACKGROUND

The travel costs associated with the locations chosen for DCMC training are a significant component of the DCMC annual training budget. The DCMC management is interested in developing a site selection model that will minimize the travel costs for personnel attending DCMC sponsored training.

OBJECTIVES

1.3.1 Review current DCMC methodology for selecting training locations.

1.3.2 Develop a site selection model using operations research techniques to minimize the travel costs of DCMC personnel.

SCOPE

Overall project functional guidance for this effort was provided by DCMC-BG, HQ DLA. All operational analysis support was accomplished by the technical support staff assigned by DORRA in Richmond, Virginia and TRAC – Monterey, Monterey, California.

ASSUMPTIONS

There are several core assumptions associated with this study. The first assumption is that any model developed must be simple for the user to use and provide useful information. The second central assumption is that the model can not cover all possible travel situations. For example, the model will not account for the traveler who commutes to the training event in a Government owned vehicle or rides as a passenger in a POV. Special cases are left up to the user. Other assumptions used: (1) All travel is done individually, i.e., no ride or room sharing. (2) Travelers will use commercial planes or POV. (3) No government meals or lodging are available. (4) Travelers with a permanent duty station (PDS) greater than 100 statute miles will fly to the training event site. Otherwise, they will drive. (5) Attendees who fly to the training event will arrive the day before the training event begins and will return to their point of origin on the last day of the training event. (6) Attendees who drive to the training event will depart the PDS the day the training event begins and return to their PDS the day the training event ends. (7) The GSA contracted airfare is available for all flights.

LITERATURE OVERVIEW

After a review of the relevant literature and an extensive search of the Internet, it was determined that no commercial software is available to solve the site selection problem for DCMC. The DCMC problem lends itself to being solved as a minimal spanning tree problem. It was not the intent of this study to evaluate every available minimal spanning algorithm, but rather to focus on a proven technique that might result in better information about the travel costs for training events. With this in mind, Dykstra's algorithm was used to solve for the least cost between all 261 airports using GSA contracted airfares to define the distances between airports.

SECTION 2 - STUDY APPROACH

EXAMPLE SOLUTION

The solution methodology used to optimize the results for this problem involves simple arithmetic and can be explained as follows. Given travelers from different cities who must come together for a meeting, the total travel cost is computed by multiplying the number of travelers from each city times the round trip transportation cost to a candidate city. The additional costs of lodging, meal and incidental expenses (M&IE) are multiplied by the number of people traveling to the training event site times the duration of the meeting in days. This can be best illustrated by example. Consider the situation where 11 people from four cities must attend a four-day meeting in St. Louis. The expense data is shown below:

Origin City	Number of Travelers	Meals and Incidental Expenses Per Day	Lodging Per Day
Atlanta	2	$38	$97
Chicago	3	$42	$120
Springfield, IL	2	$30	$55
St. Louis	4	$42	$75

Travel Costs:

Route	Round Trip Travel Cost per Person
Atlanta(ATL)-St. Louis(STL)	$204
Chicago(ORD)- St. Louis(STL)	$70
Springfield(SPI) - St. Louis(STL)	$52 (Driving Cost, 84 miles)

To hold the meeting in St. Louis, the costs are computed as follows:

- Transportation costs from all points of origin are:

(# ATL Passengers)*(ATL-STL Airfare)=(2)*(204)=$408

(# ORD Passengers)*(ORD-STL Airfare)=(3)*(70)=$210

(# SPI Passengers)*(ORD-STL Airfare)=(2)*(168 mi.*$ .31/mi)=$104

Total transportation costs are $722.

- Meals and Incidental Expenses (M&IE) are computed as per The Joint Travel Regulations allowing 75% for the first and last days traveled with 100% given for full travel days. We can calculate the St. Louis M&IE at:

(Total # of participants flying to St. Louis)*(St. Louis M&IE)*(# of days-.5(assuming the meeting is more than one day))=(5)*(42)*(4.5)=$945

(Total # of participants driving to St. Louis)*(St. Louis M&IE)*(# of days-.5(assuming the meeting is more than one day))=(2)*(42)*(3.5)=$294

- ** Note: .5 is used in the M&IE calculations to account for the 75% of first and last days M&IE allowed by the Joint Travel Regulation.
- Total M&IE expense is $1,239
- Lodging costs for passengers that fly are calculated assuming they arrive the day before the meeting and depart the day the meeting ends.

(Total # of participants who fly to STL)*(STL maximum lodging rate)*(# of days for the training event)=(5)*(75)*(4)=$1,500

- Lodging costs for passengers that drive are calculated assuming they drive to the training event the day of the meeting and depart the day the meeting ends.

(Total # of participants who drive to STL)*(STL maximum lodging rate)*(# of days for the training event - 1)=(2)*(75)*(3)=$450

Total lodging expense is $1,950

Thus the total cost to host the training event in St. Louis is:

$722 + $1,239 + $1,950 = $3,911

Computing the costs for Atlanta, Chicago, and Springfield using the same methodology yields the following results:

City	Travel	M&IE	Lodging	$ Total
Atlanta	1,768	1,539	3,492	6,799
Chicago	788	1,512	3,840	6,140
St. Louis	722	1,239	1,950	3,911
Springfield, IL	**1,034**	**1,095**	**1,760**	**3,889**

Lowest cost meeting site

It quickly becomes obvious that while this is a straight forward, albeit time consuming task, to manually calculate all of the costs, a spreadsheet would be useful for performing the calculations. However, the inherent difficulty with developing a spreadsheet based solution to this problem lie in the difficulty in obtaining all of the data required to solve the example problem.

DATA COLLECTION AND PREPROCESSING

All data points used in the model were verified by hand prior to inclusion in the model. Verification became necessary when examination of trial output revealed solution errors. It was discovered that spelling errors and abbreviations of location names and states in the source data are a problem. All six coordinates used in the latitude and longitude locations of the 261 airports and both values used in the per diem rates were verified. Once the input data were checked, the model results were verified using a combination of the General Algebraic Modeling System (GAMS) and hand calculations for test models. The OffSite results corresponded to the GAMS output and hand check of the chosen locations.

The data used for the model were collected from a variety of public sources. The General Services Agency (GSA) provided the city pair flight data at its website: http://pub.fss.gsa.gov/services/citypair.html. As previously noted, there are only 4,640 city pair GSA airfares available for 261 different airports located in the coterminous United States. The problem for the training event planner is that there remain 63,220 city pair combinations without contracted airfares.

After determining that no commercial software is available to solve for the optimal training event location, an algebraic representation of the problem was developed and is presented below.

$$\text{Min Cost} \quad \sum_{j} X_j * \left(\sum_{i} T_{ij} + P_j\right)$$

$$\text{Subject to} \sum_{j} X_j \geq 1$$

where: i,j = cities (261)

P_i = Per Diem rate at city i

T_{ij} = the cost to travel to city j from city i

X_j = 1 if meeting in city j, 0 otherwise

Dr. Dimitri P. Bertsekas of the Massachusetts Institute of Technology provided the minimal spanning tree algorithm used to solve for the least cost between airports. Dr. Bertsekas's algorithm, Relax-IV, is a schema that involves a new use of Dykstra's algorithm modified for non-capacitated shortest path network optimization. The algorithm optimizes the routes between all 261 airports in the model. The output from the algorithm is used by OffSite to calculate the flight travel costs for the training event locations chosen. The Relax-IV algorithm conducts an iterative search through all 2^{261} airport pair combinations to determine the optimal connecting route for each airport. Once the search is complete, the data are stored in an array that is used to determine the travel costs for the chosen cities. Dr. Bertsekas Relax-IV code and authorization for use in this project are on file with DORRA.

GSA publishes monthly changes to the city pair rates at http://pub.fss.gsa.gov/services/citypair.html. The rates change for a variety of reasons which can include; dropping a route or city by an airline, airline mergers, and others. However, according to Ms. Renita Nowlin of the GSA, there is normally little change with the domestic carriers that are included in this model. The contract for the city pair airfares is competed annually. Hence, the OffSite program will require annual updates. A sample of the GSA rates used in the model was compared to the "real-time" airline reservation system at the Omega World Travel office located in Richmond, Virginia. The travel agent, Ms. Heather Perez confirmed that the sample airfares were all within ± 5 % of the price quoted for 26 March 1998.

The per diem rates used for the locations in the model were provided by The Per Diem, Travel, and Transportation Allowance Committee website at: http://www.dtic.mil/perdiem/. The rates are published on an annual basis. As previously specified, the model calculations incorporate the 75% allowance for the first and last days travel of the Meals and Incidental Expenses. In the cases where seasonal per diem rates are available the higher rate is used. The lodging rate is based upon the maximum allowable rate for each area.

Driving rates are calculated using the standard $.31 cents per mile rate authorized for POV travel. The distance between airports is calculated using the latitude and longitude locations of the airports. If the distance between airports is greater than 100 miles, the program default is that the traveler will fly to the training event. If the distance is 100 miles or less the default is the calculated mileage between airports.

Rental car costs are not considered in this model for a variety of reasons. Currently, the GSA does not have contracted rates with rental car agencies. Hence the variability of rates by city and rental agency is impractical to model. Additionally, training location planners can choose to use airport hotels for their training events. Frequently airport hotels provide no cost shuttle transportation from the hotel to the airport, avoiding the cost of rental cars altogether. Discussions with Ms. Heather Perez revealed that as a rule, travelers should expect to pay an average of $45/day in rental car expenses.

The dynamics of the airline flight-scheduling system, coupled with the lack of an accounting system, prohibit any model from determining the *exact* cost for a training event. This is largely due to uncontrollable variations in potential departure and arrival locations. For example, the least cost one-way airfare from the Washington DC (BWI) airport to the CAO office in Wichita KS is $161 with travel through three airports. The same trip from Washington (National) to Wichita is $150 with travel through two airports, National and Wichita. To travel from Washington (Dulles) to Wichita is $134, with no connecting flights. The model makes its calculations based on input from the user. If the traveler normally uses BWI for departures then it will calculate his flight cost as $322. When the traveler makes his reservations, the travel agent might show the traveler that the flight is indirect and costs more than a direct flight from National or Dulles, changing his departure location. Conversely, the flights from National and Dulles might be booked, and the only available flight is the BWI flight. The model output must be considered as an estimate of the actual travel cost for the event. This is the "Catch-22" of the problem and why it is impossible to get exact costs to the most efficient meeting site. For any model to choose the most effective multiple travel routes it must know the final destination, which of course is also what this model is determining. Therefore, some basic assumptions must be made up front for the model to determine the optimal location. As previously stated the model produces reasonable and nearly optimal travel plans and destination guidelines, but there is no claim of an *exact* solution.

SECTION 3 - CONCLUSIONS

OffSite was tested using data from a recent two-day training event DCMC sponsored by DCMC at Monterey, California. Representatives from every CAO office attended the training event. The table below shows the results of the site selection model for the Monterey training event. OffSite revealed that Monterey is $10,522 more than the least cost alternative, Tulsa OK at $27,060. This represents a 39% savings on this training event alone. If OffSite is constrained to choose one of the CAO cities, then St. Louis MO is chosen at $29,896. St. Louis is 26% less expensive than Monterey.

Monterey	CA	$37,582

The 10 least expensive places in the United States:

Tulsa	OK	$27,060
Oklahoma City	OK	$27,180
Little Rock	AR	$27,622
Moline	IL	$28,114
Louisville	KY	$28,857
Reno	NV	$29,148

Ontario	CA	$29,314
Albuquerque	NM	$29,742
Lawton	OK	$29,792
St. Louis	MO	$29,896

If one origin city hosts the conference:

St. Louis	MO	$29,896
Indianapolis	IN	$30,110
Dayton	OH	$30,129
Orlando	FL	$30,466
Hartford	CT	$30,513
Cleveland	OH	$30,624
Birmingham	AL	$30,658
Albany	NY	$31,081
Wichita	KS	$31,157
Jacksonville	FL	$31,381

If DCMC could save 39% on every training event, it might realize potential $3.1 million dollar savings based upon the FY97 training budget. Even if a more conservative estimate of 5% annual savings is used, its possible that DCMC could realize an annual $400,000 saving on travel related to training.

The larger implication is that a 5% savings in the Federal training and meeting travel budget could result in as much as $100 million dollars a year in savings based upon a estimate of the annual Federal travel budget by Dennis Fischer, GSA Chief Financial Officer (www.federaltimes.com/federaltraveler/reinvent.html). These savings are achievable because all Federal Government and Department of Defense employees have access to GSA airfares, and all are subject to the per diem rates used in the calculations. This assumes widespread use of the model by meeting and training event planners. Posting OffSite on the DCMC website will provide worldwide distribution to other users.

SECTION 4 - RECOMMENDATIONS

Recommend immediate implementation of the OffSite software and funding for continued refinement of the user interface and software updates. Additionally, recommend releasing OffSite to other Federal agencies for use.

APPENDIX A

Appendix A is the source code for solving both the DCMC transportation problem and the code for the OffSite graphical user interface.

RELAX-IV MINIMAL SPANNING TREE ALGORITHM

```
C  SAMPLE CALLING PROGRAM FOR RELAX-IV
C
C------------------------------------------------------------
C
C  PURPOSE - THIS PROGRAM READS IN DATA FOR A LINEAR COST
C     ORDINARY NETWORK FLOW PROBLEM FROM THE FILE 'RELAX4.INP',
C     CALLS THE ROUTINE INIDAT TO CONSTRUCT LINKED LIST FOR THE
PROBLEM,
C     AND THEN CALLS THE ROUTINE RELAX4 TO SOLVE THE PROBLEM.
C
C------------------------------------------------------------
C
      PROGRAM MAIN
      IMPLICIT INTEGER (A-Z)
C
C     MAXNN = DIMENSION OF NODE-LENGTH ARRAYS
C     MAXNA = DIMENSION OF ARC-LENGTH ARRAYS
C
      PARAMETER (MAXNN=10000, MAXNA=70000)
C
C  INPUT PARAMETERS
C
C     N          = NUMBER OF NODES
C     NA         = NUMBER OF ARCS
C     LARGE      = A VERY LARGE INTEGER TO REPRESENT INFINITY
C     STARTN(J) = STARTING NODE FOR ARC J,          J = 1,...,NA
C     ENDN(J)   = ENDING NODE FOR ARC J,            J = 1,...,NA
C     C(J)       = COST OF ARC J,                   J = 1,...,NA
C
      INTEGER STARTN(MAXNA),ENDN(MAXNA),C(MAXNA)
      COMMON /INPUT/N,NA,LARGE
      COMMON /ARRAYS/STARTN/ARRAYE/ENDN/ARRAYC/C
C
C  UPDATED PARAMETERS
C
C     U(J)       = CAPACITY OF ARC J ON INPUT AND RESIDUAL CAPACITY
C                  ON OUTPUT,                       J = 1,...,NA
C     B(I)       = DEMAND AT NODE I ON INPUT AND ZERO ON OUTPUT,
C                                     I = 1,...,N
```

```
C
      INTEGER U(MAXNA),B(MAXNN)
      COMMON /ARRAYU/U/ARRAYB/B
C
C  OUTPUT PARAMETERS
C
C    X(J)      = FLOW ON ARC J,                    J = 1,...,NA
C    RC(J)     = REDUCED COST OF ARC J,            J = 1,...,NA
C    NMULTINODE = NUMBER OF MULTINODE RELAXATION ITERATIONS IN
RELAX4
C    ITER       = NUMBER OF RELAXATION ITERATIONS IN RELAX4
C    NUM_AUGM   = NUMBER OF FLOW AUGMENTATION STEPS IN RELAX4
C    NUM_ASCNT  = NUMBER OF MULTINODE ASCENT STEPS IN RELAX4
C    NSP        = NUMBER OF AUCTION/SHORTEST PATH ITERATIONS
C    TCOST      = COST OF FLOW
C
      INTEGER X(MAXNA),RC(MAXNA)
      REAL*8 TCOST
      COMMON /ARRAYX/X/ARRAYRC/RC
      COMMON /OUTPUT/NMULTINODE,ITER,NUM_AUGM,NUM_ASCNT,NSP
C
C  WORKING PARAMETERS
C
      INTEGER I1(MAXNN),I2(MAXNN),I3(MAXNN),I4(MAXNA)
      INTEGER I5(MAXNN),I6(MAXNA),I7(MAXNA)
      INTEGER TFSTOU(MAXNN),TNXTOU(MAXNA)
      INTEGER TFSTIN(MAXNN),TNXTIN(MAXNA)
      INTEGER I14(MAXNN),I15(MAXNN),I16(MAXNN),I17(MAXNN)
      LOGICAL*1 SCAN(MAXNN),MARK(MAXNN)
      LOGICAL*1 REPEAT
      COMMON /BLK1/I1/BLK2/I2/BLK3/I3/BLK4/I4
      COMMON /BLK5/I5/BLK6/I6/BLK7/I7
      COMMON /BLK8/SCAN/BLK9/MARK
      COMMON /BLK10/TFSTOU/BLK11/TNXTOU/BLK12/TFSTIN/BLK13/TNXTIN
      COMMON /BLK14/I14/BLK15/I15/BLK16/I16/BLK17/I17
      COMMON /CR/CRASH
      COMMON /BLKR/REPEAT
C
C  OPTIONAL WORKING PARAMETERS (FOR SENSITIVITY ANALYSIS ONLY)
C
      INTEGER CAP(MAXNA)
      COMMON /BLKCAP/CAP
C
C  DECLARE TIMING VARIABLES FOR UNIX SYSTEM
C
      REAL*4 TIME0,TIME1,TIME2
      COMMON /T/TIME0,TIME1
C
C---------------------------------------------------------------
C
C   READ PROBLEM DATA FROM FILE RELAX4.INP
C
      PRINT*,'READ PROBLEM DATA FROM RELAX4.INP'
      OPEN(13,FILE='RELAX4.INP',STATUS='OLD')
      REWIND(13)
C
C   READ NUMBER OF NODES AND ARCS
C
      READ(13,1000) N,NA
C
C   READ START NODE, END NODE, COST, AND CAPACITY OF EACH ARC
C
      DO 20 I=1,NA
        READ(13,1000) STARTN(I),ENDN(I),C(I),U(I)
20    CONTINUE
C
C   READ SUPPLY OF EACH NODE; CONVERT IT TO DEMAND
C
      DO 30 I=1,N
        READ(13,1000) B(I)
        B(I)=-B(I)
30    CONTINUE
C
1000  FORMAT(4I8)
      REWIND(13)
      CLOSE(13)

c
      PRINT*,'END OF READING'
      PRINT*,'NUMBER OF NODES =',N,', NUMBER OF ARCS =',NA
C
C   SET LARGE TO A LARGE INTEGER FOR YOUR MACHINE
C
      LARGE=500000000
      DANGER_THRESH=LARGE/10
C
C   CHECK DATA IS WITHIN RANGE OF MACHINE
C
      FLAG1=0
      FLAG2=0
      FLAG3=0
      DO 40 I=1,NA
        IF (ABS(C(I)).GT.LARGE) FLAG1=1
```

```
         IF (U(I).GT.LARGE) FLAG2=1
         IF (ABS(C(I)).GT.DANGER_THRESH) FLAG3=1
 40      CONTINUE
        IF (FLAG1.EQ.1) THEN
         PRINT*,'SOME COSTS EXCEED THE ALLOWABLE RANGE’
         PRINT*,'PROGRAM CANNOT RUN; PRESS <CR> TO EXIT’
         PAUSE
         STOP
        END IF
        IF (FLAG2.EQ.1) THEN
         PRINT*,'SOME ARC CAPACITIES EXCEED THE ALLOWABLE RANGE’
         PRINT*,'PROGRAM CANNOT RUN; PRESS <CR> TO EXIT’
         PAUSE
         STOP
        END IF
        IF (FLAG3.EQ.1) THEN
          PRINT*,'SOME COSTS ARE DANGEROUSLY LARGE’
          PRINT*,'PROGRAM MAY NOT RUN CORRECTLY’
        END IF
C
C--------------------------------------------------------------
C
C     CONSTRUCT LINKED LISTS FOR THE PROBLEM
C
        PRINT*,'CONSTRUCT LINKED LISTS FOR THE PROBLEM’

        CALL INIDAT
C
C--------------------------------------------------------------
C
C     INITIALIZE DUAL PRICES
C     (DEFAULT: ALL DUAL PRICES = 0, SO REDUCED COST IS SET
C     EQUAL TO COST)
C
        DO 60 I=1,NA
 60       RC(I)=C(I)
C
C     SPECIFY THAT WE ARE SOLVING THE PROBLEM FROM SCRATCH
C
        REPEAT=.FALSE.
C
C     STORE CAPACITY OF ARCS IN CAP
C     (OPTIONAL IF SENSITIVITY ANALYSIS WILL NOT BE ACTIVATED)
C
        DO 70 I=1,NA
 70       CAP(I)=U(I)
```

```
C
C-----------------------------------------------------------
C
C    SET CRASH EQUAL TO 1 TO ACTIVATE AN AUCTION/SHORTEST PATH
SUBROUTINE FOR
C    GETTING THE INITIAL PRICE-FLOW PAIR. THIS IS RECOMMENDED FOR
DIFFICULT
C    PROBLEMS WHERE THE DEFAULT INITIALIZATION YIELDS
C    LONG SOLUTION TIMES.
C
     PRINT*,'ENTER THE INITIALIZATION DESIRED’
     PRINT*,’<0> FOR THE DEFAULT INITIALIZATION’
     PRINT*,’<1> FOR AUCTION INITIALIZATION’
     READ*,CRASH
C
C    CALL RELAX4 TO SOLVE THE PROBLEM
C
75   CONTINUE
     PRINT*,'CALLING RELAX4 TO SOLVE THE PROBLEM’
     PRINT*,'***********************************’
C
C    INITIALIZE SYSTEM TIMER
C
C    TIME0 = LONG(362)/60.0
     TIME0 = SECNDS(0.0)
C
     CALL RELAX4
C
C    CALL SYSTEM TIMER TO DISPLAY EXECUTION TIME FOR RELAX4
C
C     TIME2 = LONG(362)/60.0 - TIME0
     TIME2 = SECNDS(TIME0)

     PRINT*,'TOTAL SOLUTION TIME =',TIME2,’ SECS.’
     PRINT*,'TIME IN INITIALIZATION =',TIME1,’ SECS.’
C
C-----------------------------------------------------------
C
C    CHECK CORRECTNESS OF OUTPUT PARAMETERS
C
     DO 80 NODE=1,N
       IF (B(NODE).NE.0) THEN
         PRINT*,'NONZERO SURPLUS AT NODE ',NODE
       END IF
80   CONTINUE
     DO 90 ARC=1,NA
```

```
        IF (X(ARC).GT.0) THEN
         IF (RC(ARC).GT.0) THEN
          PRINT*,'COMPLEMENTARY SLACKNESS VIOLATED AT ARC ',ARC
         ENDIF
        ENDIF
        IF (U(ARC).GT.0) THEN
         IF (RC(ARC).LT.0) THEN
          PRINT*,'COMPLEMENTARY SLACKNESS VIOLATED AT ARC ',ARC
         ENDIF
        ENDIF
90    CONTINUE
C
C     COMPUTE AND DISPLAY COST OF FLOWS (IN DOUBLE PRECISION)
C
      TCOST=FLOAT(0)
      DO 100 I=1,NA
       TCOST=TCOST + DFLOAT(X(I)*C(I))
100   CONTINUE
      PRINT*,'OPTIMAL COST = ',TCOST
C
C     DISPLAY RELAX4 STATISTICS
C
      IF (CRASH.EQ.1) THEN
       PRINT*,'NUMBER OF AUCTION/SHORTEST PATH ITERATIONS =',NSP
      END IF
      PRINT*,'NUMBER OF ITERATIONS = ',ITER
      PRINT*,'NUMBER OF MULTINODE ITERATIONS = ',NMULTINODE
      PRINT*,'NUMBER OF MULTINODE ASCENT STEPS = ',NUM_ASCNT
      PRINT*,'NUMBER OF REGULAR AUGMENTATIONS = ',NUM_AUGM
      PRINT*,'***********************************'
C
C     TO ACTIVATE SENSITIVITY ANALYSIS, INSERT THE FOLLOWING
C     THREE LINES HERE.
C
      CALL SENSTV
      REPEAT=.TRUE.
      GO TO 75
C
      END
C
C
      SUBROUTINE INIDAT
      IMPLICIT INTEGER (A-Z)
C
C--------------------------------------------------------------
C
```

```
C  PURPOSE - THIS ROUTINE CONSTRUCTS TWO LINKED LISTS FOR
C    THE NETWORK TOPOLOGY: ONE LIST (GIVEN BY FOU, NXTOU) FOR
C    THE OUTGOING ARCS OF NODES AND ONE LIST (GIVEN BY FIN,
C    NXTIN) FOR THE INCOMING ARCS OF NODES.  THESE TWO LISTS
C    ARE REQUIRED BY RELAX4.
C
C-------------------------------------------------------------
C
C    MAXNN = DIMENSION OF NODE-LENGTH ARRAYS
C    MAXNA = DIMENSION OF ARC-LENGTH ARRAYS
C
      PARAMETER (MAXNN=10000, MAXNA=70000)
C
C  INPUT PARAMETERS
C
C    N         = NUMBER OF NODES
C    NA        = NUMBER OF ARCS
C    STARTN(J) = STARTING NODE FOR ARC J,          J = 1,...,NA
C    ENDN(J)   = ENDING NODE FOR ARC J,            J = 1,...,NA
C
      INTEGER STARTN(MAXNA),ENDN(MAXNA)
      COMMON /ARRAYS/STARTN/ARRAYE/ENDN
      COMMON /INPUT/N,NA,LARGE
C
C  OUTPUT PARAMETERS
C
C    FOU(I)    = FIRST ARC OUT OF NODE I,          I = 1,...,N
C    NXTOU(J)  = NEXT ARC OUT OF THE STARTING NODE OF ARC J,
C                                  J = 1,...,NA
C    FIN(I)    = FIRST ARC INTO NODE I,            I = 1,...,N
C    NXTIN(J)  = NEXT ARC INTO THE ENDING NODE OF ARC J,
C                                  J = 1,...,NA
C
      INTEGER FOU(MAXNN),NXTOU(MAXNA),FIN(MAXNN),NXTIN(MAXNA)
      COMMON /BLK3/FOU/BLK4/NXTOU/BLK5/FIN/BLK6/NXTIN
C
C  WORKING PARAMETERS
C
      INTEGER TEMPIN(MAXNN),TEMPOU(MAXNN)
      COMMON /BLK1/TEMPIN/BLK2/TEMPOU
C
C-------------------------------------------------------------
C
      DO 10 I=1,N
        FIN(I)=0
        FOU(I)=0
```

```
         TEMPIN(I)=0
 10       TEMPOU(I)=0
        DO 20 I=1,NA
         NXTIN(I)=0
         NXTOU(I)=0
         I1=STARTN(I)
         I2=ENDN(I)
         IF (FOU(I1).NE.0) THEN
          NXTOU(TEMPOU(I1))=I
         ELSE
          FOU(I1)=I
         END IF
         TEMPOU(I1)=I
         IF (FIN(I2).NE.0) THEN
          NXTIN(TEMPIN(I2))=I
         ELSE
          FIN(I2)=I
         END IF
 20       TEMPIN(I2)=I
        RETURN
        END
C
C
        SUBROUTINE RELAX4
        IMPLICIT INTEGER (A-Z)
C
C--------------------------------------------------------------
C
C             RELAX-IV  (VERSION OF OCTOBER 1994)
C
C  RELEASE NOTE - THIS VERSION OF RELAXATION CODE HAS OPTION FOR
C    A SPECIAL CRASH PROCEDURE FOR
C    THE INITIAL PRICE-FLOW PAIR. THIS IS RECOMMENDED FOR
DIFFICULT
C    PROBLEMS WHERE THE DEFAULT INITIALIZATION
C    RESULTS IN LONG RUNNING TIMES.
C    CRASH =1 CORRESPONDS TO AN AUCTION/SHORTEST PATH METHOD
C
C    THESE INITIALIZATIONS ARE RECOMMENDED IN THE ABSENCE OF
ANY
C    PRIOR INFORMATION ON A FAVORABLE INITIAL FLOW-PRICE VECTOR
PAIR
C    THAT SATISFIES COMPLEMENTARY SLACKNESS
C
C    THE RELAXATION PORTION OF THE CODE DIFFERS FROM THE CODE
RELAXT-III
```

```
C    AND OTHER EARLIER RELAXATION CODES IN THAT IT MAINTAINS
C    THE SET OF NODES WITH NONZERO DEFICIT IN A FIFO QUEUE.
C    LIKE ITS PREDECESSOR RELAXT-III, THIS CODE MAINTAINS A LINKED
LIST
C    OF BALANCED (I.E., OF ZERO REDUCED COST) ARCS SO TO REDUCE
C    THE WORK IN LABELING AND SCANNING.
C    UNLIKE RELAXT-III, IT DOES NOT USE SELECTIVELY
C    SHORTEST PATH ITERATIONS FOR INITIALIZATION.
C
C-----------------------------------------------------------
C
C  PURPOSE - THIS ROUTINE IMPLEMENTS THE RELAXATION METHOD
C    OF BERTSEKAS AND TSENG (SEE [1], [2]) FOR LINEAR
C    COST ORDINARY NETWORK FLOW PROBLEMS.
C
C  [1] BERTSEKAS, D. P., "A UNIFIED FRAMEWORK FOR PRIMAL-DUAL
METHODS ..."
C     MATHEMATICAL PROGRAMMING, VOL. 32, 1985, PP. 125-145.
C  [2] BERTSEKAS, D. P., AND TSENG, P., "RELAXATION METHODS FOR
C     MINIMUM COST ..." OPERATIONS RESEARCH, VOL. 26, 1988, PP. 93-114.
C
C    THE RELAXATION METHOD IS ALSO DESCRIBED IN THE BOOKS:
C
C  [3] BERTSEKAS, D. P., "LINEAR NETWORK OPTIMIZATION: ALGORITHMS
AND CODES"
C     MIT PRESS, 1991.
C  [4] BERTSEKAS, D. P. AND TSITSIKLIS, J. N., "PARALLEL AND
DISTRIBUTED
C     COMPUTATION: NUMERICAL METHODS", PRENTICE-HALL, 1989.
C
C
C
C-----------------------------------------------------------
C
C  SOURCE -  THIS CODE WAS WRITTEN BY DIMITRI P. BERTSEKAS
C    AND PAUL TSENG, WITH A CONTRIBUTION BY JONATHAN ECKSTEIN
C    IN THE PHASE II INITIALIZATION.  THE ROUTINE AUCTION WAS
WRITTEN
C    BY DIMITRI P. BERTSEKAS AND IS BASED ON THE METHOD DESCRIBED
IN
C    THE PAPER:
C
C  [5] BERTSEKAS, D. P., "AN AUCTION/SEQUENTIAL SHORTEST PATH
ALGORITHM
C    FOR THE MINIMUM COST FLOW PROBLEM", LIDS REPORT P-2146, MIT,
NOV. 1992.
```

```
C
C    FOR INQUIRIES ABOUT THE CODE, PLEASE CONTACT:
C
C    DIMITRI P. BERTSEKAS
C    LABORATORY FOR INFORMATION AND DECISION SYSTEMS
C    MASSACHUSETTS INSTITUTE OF TECHNOLOGY
C    CAMBRIDGE, MA 02139
C    (617) 253-7267, DIMITRIB@MIT.EDU
C
C-----------------------------------------------------------
C
C  USER GUIDELINES -
C
C    THIS ROUTINE IS IN THE PUBLIC DOMAIN TO BE USED ONLY FOR
RESEARCH
C    PURPOSES.  IT CANNOT BE USED AS PART OF A COMMERCIAL
PRODUCT, OR
C    TO SATISFY IN ANY PART COMMERCIAL DELIVERY REQUIREMENTS TO
C    GOVERNMENT OR INDUSTRY, WITHOUT PRIOR AGREEMENT WITH THE
AUTHORS.
C    USERS ARE REQUESTED TO ACKNOWLEDGE THE AUTHORSHIP OF THE
CODE,
C    AND THE RELAXATION METHOD.  THEY SHOULD ALSO REGISTER WITH
THE
C    AUTHORS TO RECEIVE UPDATES AND SUBSEQUENT RELEASES.
C
C    NO MODIFICATION SHOULD BE MADE TO THIS CODE OTHER
C    THAN THE MINIMAL NECESSARY
C    TO MAKE IT COMPATIBLE WITH THE FORTRAN COMPILERS OF
SPECIFIC
C    MACHINES.  WHEN REPORTING COMPUTATIONAL RESULTS PLEASE BE
SURE
C    TO DESCRIBE THE MEMORY LIMITATIONS OF YOUR MACHINE.
GENERALLY
C    RELAX4 REQUIRES MORE MEMORY THAN PRIMAL SIMPLEX CODES
AND MAY
C    BE PENALIZED SEVERELY BY LIMITED MACHINE MEMORY.
C
C-----------------------------------------------------------
C
C    MAXNN = DIMENSION OF NODE-LENGTH ARRAYS
C    MAXNA = DIMENSION OF ARC-LENGTH ARRAYS
C
     PARAMETER (MAXNN=10000, MAXNA=70000)
C
C  INPUT PARAMETERS (SEE NOTES 1, 2, 4)
```

```
C
C   N         = NUMBER OF NODES
C   NA        = NUMBER OF ARCS
C   LARGE     = A VERY LARGE INTEGER TO REPRESENT INFINITY
C               (SEE NOTE 3)
C   REPEAT    = .TRUE. IF INITIALIZATION IS TO BE SKIPPED
C               (.FALSE. OTHERWISE)
C   CRASH     = 0 IF DEFAULT INITIALIZATION IS USED
C               1 IF AUCTION INITIALIZATION IS USED
C   STARTN(J) = STARTING NODE FOR ARC J,       J = 1,...,NA
C   ENDN(J)   = ENDING NODE FOR ARC J,         J = 1,...,NA
C   FOU(I)    = FIRST ARC OUT OF NODE I,       I = 1,...,N
C   NXTOU(J)  = NEXT ARC OUT OF THE STARTING NODE OF ARC J,
C                                J = 1,...,NA
C   FIN(I)    = FIRST ARC INTO NODE I,         I = 1,...,N
C   NXTIN(J)  = NEXT ARC INTO THE ENDING NODE OF ARC J,
C                                J = 1,...,NA
C
      INTEGER STARTN(MAXNA),ENDN(MAXNA)
      INTEGER FOU(MAXNN),NXTOU(MAXNA),FIN(MAXNN),NXTIN(MAXNA)
      LOGICAL*1 REPEAT
      COMMON /INPUT/N,NA,LARGE
      COMMON /ARRAYS/STARTN/ARRAYE/ENDN
      COMMON /BLK3/FOU/BLK4/NXTOU/BLK5/FIN/BLK6/NXTIN
      COMMON /BLKR/REPEAT
      COMMON /CR/CRASH
C
C UPDATED PARAMETERS (SEE NOTES 1, 3, 4)
C
C   RC(J)     = REDUCED COST OF ARC J,         J = 1,...,NA
C   U(J)      = CAPACITY OF ARC J ON INPUT
C               AND (CAPACITY OF ARC J) - X(J) ON OUTPUT,
C                                J = 1,...,NA
C   DFCT(I)   = DEMAND AT NODE I ON INPUT
C               AND ZERO ON OUTPUT,            I = 1,...,N
C
      INTEGER RC(MAXNA),U(MAXNA),DFCT(MAXNN)
      COMMON /ARRAYRC/RC/ARRAYU/U/ARRAYB/DFCT
C
C OUTPUT PARAMETERS (SEE NOTES 1, 3, 4)
C
C   X(J)      = FLOW ON ARC J,                 J = 1,...,NA
C   NMULTINODE = NUMBER OF MULTINODE RELAXATION ITERATIONS IN
RELAX4
C   ITER      = NUMBER OF RELAXATION ITERATIONS IN RELAX4
C   NUM_AUGM  = NUMBER OF FLOW AUGMENTATION STEPS IN RELAX4
```

```
C    NUM_ASCNT  = NUMBER OF MULTINODE ASCENT STEPS IN RELAX4
C    NSP        = NUMBER OF AUCTION/SHORTEST PATH ITERATIONS
C
     INTEGER X(MAXNA)
     COMMON /ARRAYX/X
     COMMON/OUTPUT/NMULTINODE,ITER,NUM_AUGM,NUM_ASCNT,NSP
C
C  WORKING PARAMETERS (SEE NOTES 1, 4, 5)
C
     INTEGER LABEL(MAXNN),PRDCSR(MAXNN),SAVE(MAXNA)
     INTEGER
TFSTOU(MAXNN),TNXTOU(MAXNA),TFSTIN(MAXNN),TNXTIN(MAXNA)
     INTEGER DDPOS(MAXNN),DDNEG(MAXNN),NXTQUEUE(MAXNN)
     INTEGER I15(MAXNN),I16(MAXNN),I17(MAXNN)
     LOGICAL*1 SCAN(MAXNN),MARK(MAXNN)
     LOGICAL*1 FEASBL,QUIT,SWITCH,POSIT,PCHANGE
     COMMON /BLK1/LABEL/BLK2/PRDCSR/BLK7/SAVE
     COMMON /BLK10/TFSTOU/BLK11/TNXTOU/BLK12/TFSTIN/BLK13/TNXTIN
     COMMON /BLK14/NXTQUEUE/BLK15/I15/BLK16/I16/BLK17/I17
     COMMON /BLK8/SCAN/BLK9/MARK
     EQUIVALENCE(DDPOS,TFSTOU),(DDNEG,TFSTIN)
C
C  TIMING PARAMETERS
C
     REAL*4 TIME0,TIME1
     COMMON /T/TIME0,TIME1
C
C  NOTE 1 -
C    TO RUN IN LIMITED MEMORY SYSTEMS, DECLARE THE ARRAYS
C    STARTN, ENDN, NXTIN, NXTOU, FIN, FOU, LABEL,
C    PRDCSR, SAVE, TFSTOU, TNXTOU, TFSTIN, TNXTIN,
C    DDPOS,DDNEG,NXTQUEUE AS INTEGER*2 INSTEAD.
C
C  NOTE 2 -
C    THIS ROUTINE MAKES NO EFFORT TO INITIALIZE WITH A FAVORABLE
X
C    FROM AMONGST THOSE FLOW VECTORS THAT SATISFY
COMPLEMENTARY SLACKNESS
C    WITH THE INITIAL REDUCED COST VECTOR RC.
C    IF A FAVORABLE X IS KNOWN, THEN IT CAN BE PASSED, TOGETHER
C    WITH THE CORRESPONDING ARRAYS U AND DFCT, TO THIS ROUTINE
C    DIRECTLY.  THIS, HOWEVER, REQUIRES THAT THE CAPACITY
C    TIGHTENING PORTION AND THE FLOW INITIALIZATION PORTION
C    OF THIS ROUTINE (UP TO LINE LABELED 90) BE SKIPPED.
C
C  NOTE 3 -
```

```
C    ALL PROBLEM DATA SHOULD BE LESS THAN LARGE IN MAGNITUDE,
C    AND LARGE SHOULD BE LESS THAN, SAY, 1/4 THE LARGEST INTEGER*4
C    OF THE MACHINE USED.  THIS WILL GUARD PRIMARILY AGAINST
C    OVERFLOW IN UNCAPACITATED PROBLEMS WHERE THE ARC
CAPACITIES
C    ARE TAKEN FINITE BUT VERY LARGE.  NOTE, HOWEVER, THAT AS IN
C    ALL CODES OPERATING WITH INTEGERS, OVERFLOW MAY OCCUR IF
SOME
C    OF THE PROBLEM DATA TAKES VERY LARGE VALUES.
C
C  NOTE 4 -
C    EACH COMMON BLOCK CONTAINS JUST ONE ARRAY, SO THE ARRAYS
IN RELAX4
C    CAN BE DIMENSIONED TO 1 AND TAKE THEIR DIMENSION FROM THE
C    MAIN CALLING ROUTINE.  WITH THIS TRICK, RELAX4 NEED NOT BE
RECOMPILED
C    IF THE ARRAY DIMENSIONS IN THE CALLING ROUTINE CHANGE.
C    IF YOUR FORTRAN COMPILER DOES NOT SUPPORT THIS FEATURE,
THEN
C    CHANGE THE DIMENSION OF ALL THE ARRAYS TO BE THE SAME AS
THE ONES
C    DECLARED IN THE MAIN CALLING PROGRAM.
C
C  NOTE 5 -
C    DDPOS AND DDNEG ARE ARRAYS THAT GIVE THE DIRECTIONAL
DERIVATIVES FOR
C    ALL POSITIVE AND NEGATIVE SINGLE-NODE PRICE CHANGES.  THESE
ARE USED
C    ONLY IN PHASE II OF THE INITIALIZATION PROCEDURE, BEFORE THE
C    LINKED LIST OF BALANCED ARCS COMES TO PLAY.  THEREFORE, TO
REDUCE
C    STORAGE, THEY ARE EQUIVALENCE TO TFSTOU AND TFSTIN,
C    WHICH ARE OF THE SAME SIZE (NUMBER OF NODES) AND ARE USED
C    ONLY AFTER THE TREE COMES INTO USE.
C
C---------------------------------------------------------------
C
C  INITIALIZATION PHASE I
C
C    IN THIS PHASE, WE REDUCE THE ARC CAPACITIES BY AS MUCH AS
C    POSSIBLE WITHOUT CHANGING THE PROBLEM;
C    THEN WE SET THE INITIAL FLOW ARRAY X, TOGETHER WITH
C    THE CORRESPONDING ARRAYS U AND DFCT.
C
C    THIS PHASE AND PHASE II (FROM HERE UP TO LINE LABELED 90)
```

```
C    CAN BE SKIPPED (BY SETTING REPEAT TO .TRUE.) IF THE CALLING
PROGRAM
C    PLACES IN COMMON USER-CHOSEN VALUES FOR THE ARC FLOWS, THE
RESIDUAL ARC
C    CAPACITIES, AND THE NODAL DEFICITS.  WHEN THIS IS DONE,
C    IT IS CRITICAL THAT THE FLOW AND THE REDUCED COST FOR EACH
ARC
C    SATISFY COMPLEMENTARY SLACKNESS
C    AND THE DFCT ARRAY PROPERLY CORRESPOND TO THE INITIAL
ARC/FLOWS.
C
   IF (REPEAT) GO TO 90
C
   DO 10 NODE=1,N

    NODE_DEF=DFCT(NODE)
    DDPOS(NODE)=NODE_DEF
    DDNEG(NODE)=-NODE_DEF
    MAXCAP=0
    SCAPOU=0
    ARC=FOU(NODE)
11     IF (ARC.GT.0) THEN
     IF (SCAPOU.LE.LARGE-U(ARC)) THEN
      SCAPOU=SCAPOU+U(ARC)
     ELSE
      GO TO 10
     END IF
     ARC=NXTOU(ARC)
     GO TO 11
    END IF
    IF (SCAPOU.LE.LARGE-NODE_DEF) THEN
     CAPOUT=SCAPOU+NODE_DEF
    ELSE
     GO TO 10
    END IF
    IF (CAPOUT.LT.0) THEN
C
C    PROBLEM IS INFEASIBLE - EXIT
C
     PRINT*,'EXIT DURING CAPACITY ADJUSTMENT’
     PRINT*,'EXOGENOUS FLOW INTO NODE’,NODE,
  $    ’EXCEEDS OUT CAPACITY’
     CALL PRINTFLOWS(NODE)
     GO TO 4400
    END IF
C
```

```
      SCAPIN=0
      ARC=FIN(NODE)
 12    IF (ARC.GT.0) THEN
       U(ARC)=MIN0(U(ARC),CAPOUT)
       IF (MAXCAP.LT.U(ARC)) MAXCAP=U(ARC)
       IF (SCAPIN.LE.LARGE-U(ARC)) THEN
        SCAPIN=SCAPIN+U(ARC)
       ELSE
        GO TO 10
       END IF
       ARC=NXTIN(ARC)
       GO TO 12
      END IF
      IF (SCAPIN.LE.LARGE+NODE_DEF) THEN
       CAPIN=SCAPIN-NODE_DEF
      ELSE
       GO TO 10
      END IF
      IF (CAPIN.LT.0) THEN
C
C    PROBLEM IS INFEASIBLE - EXIT
C
       PRINT*,'EXIT DURING CAPACITY ADJUSTMENT'
       PRINT*,'EXOGENOUS FLOW OUT OF NODE',NODE,
     $  'EXCEEDS IN CAPACITY'
       CALL PRINTFLOWS(NODE)
       GO TO 4400
      END IF
C
      ARC=FOU(NODE)
 15    IF (ARC.GT.0) THEN
       U(ARC)=MIN0(U(ARC),CAPIN)
       ARC=NXTOU(ARC)
       GO TO 15
      END IF
 10   CONTINUE
C
C--------------------------------------------------------------
C
C  INITIALIZATION PHASE II
C
C    IN THIS PHASE, WE INITIALIZE THE PRICES AND FLOWS BY EITHER
CALLING
C    THE ROUTINE AUCTION OR BY PERFORMING ONLY SINGLE NODE
(COORDINATE)
C    RELAXATION ITERATIONS.
```

```
C
      IF (CRASH.EQ.1) THEN
       NSP=0
       CALL AUCTION
       GO TO 70
      END IF
C
C     INITIALIZE THE ARC FLOWS TO SATISFY COMPLEMENTARY
SLACKNESS WITH THE
C     PRICES.  U(ARC) IS THE RESIDUAL CAPACITY OF ARC, AND X(ARC) IS
THE FLOW.
C     THESE TWO ALWAYS ADD UP TO THE TOTAL CAPACITY FOR ARC.
C     ALSO COMPUTE THE DIRECTIONAL DERIVATIVES FOR EACH
COORDINATE
C     AND COMPUTE THE ACTUAL DEFICITS.
C
      DO 20 ARC=1,NA
       X(ARC) = 0
       IF (RC(ARC).LE. 0) THEN
        T  = U(ARC)
        T1 = STARTN(ARC)
        T2 = ENDN(ARC)
        DDPOS(T1) = DDPOS(T1) + T
        DDNEG(T2) = DDNEG(T2) + T
        IF (RC(ARC).LT. 0) THEN
         X(ARC) = T
         U(ARC) = 0
         DFCT(T1) = DFCT(T1) + T
         DFCT(T2) = DFCT(T2) - T
         DDNEG(T1) = DDNEG(T1) - T
         DDPOS(T2) = DDPOS(T2) - T
        END IF
       END IF
20     CONTINUE
C
C     MAKE 2 OR 3 PASSES THROUGH ALL NODES, PERFORMING ONLY
C     SINGLE NODE RELAXATION ITERATIONS.  THE NUMBER OF
C     PASSES DEPENDS ON THE DENSITY OF THE NETWORK
C
      IF (NA.GT.N*10) THEN
       NUMPASSES=2
      ELSE
       NUMPASSES=3
      END IF
C
      DO 30 PASSES = 1,NUMPASSES
      DO 40 NODE=1,N

       IF (DFCT(NODE).EQ. 0) GO TO 40

       IF (DDPOS(NODE).LE. 0) THEN
C
C     COMPUTE DELPRC, THE STEPSIZE TO THE NEXT BREAKPOINT
C     IN THE DUAL COST AS THE PRICE OF NODE IS INCREASED.
C     [SINCE THE REDUCED COST OF ALL OUTGOING (RESP.,
C     INCOMING) ARCS WILL DECREASE (RESP., INCREASE) AS
C     THE PRICE OF NODE IS INCREASED, THE NEXT BREAKPOINT IS
C     THE MINIMUM OF THE POSITIVE REDUCED COST ON OUTGOING
C     ARCS AND OF THE NEGATIVE REDUCED COST ON INCOMING ARCS.]
C
        DELPRC = LARGE
        ARC = FOU(NODE)
51       IF (ARC.GT.0) THEN
         TRC = RC(ARC)
         IF ((TRC.GT. 0).AND.(TRC.LT.DELPRC)) THEN
          DELPRC = TRC
         END IF
         ARC = NXTOU(ARC)
         GOTO 51
        END IF
        ARC = FIN(NODE)
52       IF (ARC.GT.0) THEN
         TRC = RC(ARC)
         IF ((TRC.LT.0).AND.(TRC.GT.-DELPRC)) THEN
          DELPRC = -TRC
         END IF
         ARC = NXTIN(ARC)
         GOTO 52
        END IF
C
C     IF NO BREAKPOINT IS LEFT AND DUAL ASCENT IS STILL
C     POSSIBLE, THE PROBLEM IS INFEASIBLE.
C
       IF (DELPRC.GE.LARGE) THEN
        IF (DDPOS(NODE).EQ.0) GOTO 40
        GOTO 4400
       END IF
C
C     DELPRC IS THE STEPSIZE TO NEXT BREAKPOINT.  INCREASE
C     PRICE OF NODE BY DELPRC AND COMPUTE THE STEPSIZE TO
C     THE NEXT BREAKPOINT IN THE DUAL COST.
```

```
C
53     NXTBRK = LARGE
C
C   LOOK AT ALL ARCS OUT OF NODE.
C
      ARC = FOU(NODE)
54     IF (ARC.GT.0) THEN
       TRC = RC(ARC)
       IF (TRC .EQ. 0) THEN
        T1 = ENDN(ARC)
        T  = U(ARC)
        IF (T.GT.0) THEN
         DFCT(NODE) = DFCT(NODE) + T
         DFCT(T1) = DFCT(T1) - T
         X(ARC) = T
         U(ARC) = 0
        ELSE
         T = X(ARC)
        END IF
        DDNEG(NODE) = DDNEG(NODE) - T
        DDPOS(T1) = DDPOS(T1) - T
       END IF
C
C   DECREASE THE REDUCED COST ON ALL OUTGOING ARCS.
C
       TRC = TRC - DELPRC
       IF ((TRC.GT.0).AND.(TRC.LT.NXTBRK)) THEN
        NXTBRK = TRC
       ELSE IF (TRC.EQ.0) THEN
C
C   ARC GOES FROM INACTIVE TO BALANCED.  UPDATE THE
C   RATE OF DUAL ASCENT AT NODE AND AT ITS NEIGHBOR.
C
        DDPOS(NODE) = DDPOS(NODE) + U(ARC)
        DDNEG(ENDN(ARC)) = DDNEG(ENDN(ARC)) + U(ARC)
       END IF
       RC(ARC) = TRC
       ARC =  NXTOU(ARC)
       GOTO 54
      END IF
C
C   LOOK AT ALL ARCS INTO NODE.
C
      ARC = FIN(NODE)
55     IF (ARC.GT.0) THEN
       TRC = RC(ARC)
```

```
             IF (TRC.EQ.0) THEN
               T1 = STARTN(ARC)
               T  = X(ARC)
               IF (T.GT.0) THEN
                 DFCT(NODE) = DFCT(NODE) + T
                 DFCT(T1) = DFCT(T1) - T
                 U(ARC) = T
                 X(ARC) = 0
               ELSE
                 T = U(ARC)
               END IF
               DDPOS(T1) = DDPOS(T1) - T
               DDNEG(NODE) = DDNEG(NODE) - T
             END IF
C
C       INCREASE THE REDUCED COST ON ALL INCOMING ARCS.
C
             TRC = TRC + DELPRC
             IF ((TRC.LT.0).AND.(TRC.GT.-NXTBRK)) THEN
               NXTBRK = -TRC
             ELSE IF (TRC.EQ.0) THEN
C
C       ARC GOES FROM ACTIVE TO BALANCED.  UPDATE THE
C       RATE OF DUAL ASCENT AT NODE AND AT ITS NEIGHBOR.
C
               DDNEG(STARTN(ARC)) = DDNEG(STARTN(ARC)) + X(ARC)
               DDPOS(NODE) = DDPOS(NODE) + X(ARC)
             END IF
             RC(ARC) = TRC
             ARC = NXTIN(ARC)
             GOTO 55
           END IF
C
C       IF PRICE OF NODE CAN BE INCREASED FURTHER WITHOUT
DECREASING
C       THE DUAL COST (EVEN IF THE DUAL COST DOESN'T INCREASE),
C       RETURN TO INCREASE THE PRICE FURTHER.
C
          IF ((DDPOS(NODE).LE.0).AND.(NXTBRK.LT.LARGE)) THEN
            DELPRC = NXTBRK
            GOTO 53
          END IF

        ELSE IF (DDNEG(NODE).LE.0) THEN
C
C       COMPUTE DELPRC, THE STEPSIZE TO THE NEXT BREAKPOINT
```

```
C    IN THE DUAL COST AS THE PRICE OF NODE IS DECREASED.
C    [SINCE THE REDUCED COST OF ALL OUTGOING (RESP.,
C    INCOMING) ARCS WILL INCREASE (RESP., DECREASE) AS
C    THE PRICE OF NODE IS DECREASED, THE NEXT BREAKPOINT IS
C    THE MINIMUM OF THE NEGATIVE REDUCED COST ON OUTGOING
C    ARCS AND OF THE POSITIVE REDUCED COST ON INCOMING ARCS.]
C
      DELPRC = LARGE
      ARC = FOU(NODE)
61      IF (ARC.GT.0) THEN
        TRC = RC(ARC)
        IF ((TRC.LT.0).AND.(TRC.GT.-DELPRC)) THEN
          DELPRC = -TRC
        ENDIF
        ARC = NXTOU(ARC)
        GOTO 61
      END IF
      ARC = FIN(NODE)
62      IF (ARC.GT.0) THEN
        TRC = RC(ARC)
        IF ((TRC.GT.0).AND.(TRC.LT.DELPRC)) THEN
          DELPRC = TRC
        END IF
        ARC = NXTIN(ARC)
        GOTO 62
      END IF
C
C    IF NO BREAKPOINT IS LEFT AND DUAL ASCENT IS STILL
C    POSSIBLE, THE PROBLEM IS INFEASIBLE.
C
      IF (DELPRC.EQ.LARGE) THEN
        IF (DDNEG(NODE).EQ.0) GOTO 40
        GOTO 4400
      END IF
C
C    DELPRC IS THE STEPSIZE TO NEXT BREAKPOINT.  DECREASE
C    PRICE OF NODE BY DELPRC AND COMPUTE THE STEPSIZE TO
C    THE NEXT BREAKPOINT IN THE DUAL COST.
C
63      NXTBRK = LARGE
C
C    LOOK AT ALL ARCS OUT OF NODE.
C
      ARC = FOU(NODE)
64      IF (ARC.GT.0) THEN
        TRC = RC(ARC)
        IF (TRC.EQ.0) THEN
         T1 = ENDN(ARC)
         T  = X(ARC)
         IF (T.GT.0) THEN
          DFCT(NODE) = DFCT(NODE) - T
          DFCT(T1) = DFCT(T1) + T
          U(ARC) = T
          X(ARC) = 0
         ELSE
          T = U(ARC)
         END IF
         DDPOS(NODE) = DDPOS(NODE) - T
         DDNEG(T1) = DDNEG(T1) - T
        END IF
C
C   INCREASE THE REDUCED COST ON ALL OUTGOING ARCS.
C
        TRC = TRC + DELPRC
        IF ((TRC.LT.0).AND.(TRC.GT.-NXTBRK)) THEN
         NXTBRK = -TRC
        ELSE IF (TRC.EQ.0) THEN
C
C   ARC GOES FROM ACTIVE TO BALANCED.  UPDATE THE
C   RATE OF DUAL ASCENT AT NODE AND AT ITS NEIGHBOR.
C
         DDNEG(NODE) = DDNEG(NODE) + X(ARC)
         DDPOS(ENDN(ARC)) = DDPOS(ENDN(ARC)) + X(ARC)
        END IF
        RC(ARC) = TRC
        ARC = NXTOU(ARC)
        GOTO 64
       END IF
C
C   LOOK AT ALL ARCS INTO NODE.
C
      ARC = FIN(NODE)
65     IF (ARC.GT.0) THEN
       TRC = RC(ARC)
       IF (TRC.EQ.0) THEN
        T1 = STARTN(ARC)
        T  = U(ARC)
        IF (T.GT.0) THEN
         DFCT(NODE) = DFCT(NODE) - T
         DFCT(T1) = DFCT(T1) + T
         X(ARC) = T
         U(ARC) = 0
```

```
           ELSE
            T = X(ARC)
           END IF
           DDNEG(T1) = DDNEG(T1) - T
           DDPOS(NODE) = DDPOS(NODE) - T
          END IF
C
C    DECREASE THE REDUCED COST ON ALL INCOMING ARCS.
C
          TRC = TRC - DELPRC
          IF ((TRC.GT.0).AND.(TRC.LT.NXTBRK)) THEN
           NXTBRK = TRC
          ELSE IF (TRC.EQ.0) THEN
C
C    ARC GOES FROM INACTIVE TO BALANCED.  UPDATE THE
C    RATE OF DUAL ASCENT AT NODE AND AT ITS NEIGHBOR.
C
           DDPOS(STARTN(ARC)) = DDPOS(STARTN(ARC)) + U(ARC)
           DDNEG(NODE) = DDNEG(NODE) + U(ARC)
          END IF
          RC(ARC) = TRC
          ARC = NXTIN(ARC)
          GOTO 65
         END IF
C
C    IF PRICE OF NODE CAN BE DECREASED FURTHER WITHOUT
DECREASING
C    THE DUAL COST (EVEN IF THE DUAL COST DOESNT INCREASE),
C    RETURN TO DECREASE THE PRICE FURTHER.
C
        IF ((DDNEG(NODE).LE.0).AND.(NXTBRK.LT.LARGE)) THEN
         DELPRC = NXTBRK
         GOTO 63
        END IF

      END IF

40    CONTINUE

30    CONTINUE
C
C
70    CONTINUE
C
C   READ TIME FOR INITIALIZATION
C
```

```
C     TIME1 = LONG(362)/60.0 - TIME0
      TIME1 = SECNDS(TIME0)
C
C-----------------------------------------------------------
C
C     INITIALIZE TREE DATA STRUCTURE.

      DO 80 I=1,N
        TFSTOU(I)=0
        TFSTIN(I)=0
80    CONTINUE
      DO 81 I=1,NA
        TNXTIN(I)=-1
        TNXTOU(I)=-1
        IF (RC(I).EQ.0) THEN
          TNXTOU(I)=TFSTOU(STARTN(I))
          TFSTOU(STARTN(I))=I
          TNXTIN(I)=TFSTIN(ENDN(I))
          TFSTIN(ENDN(I))=I
        END IF
81    CONTINUE

C
C     INITIALIZE OTHER VARIABLES.
C
90    FEASBL=.TRUE.
      ITER=0
      NMULTINODE=0
      NUM_AUGM=0
      NUM_ASCNT=0
      NUM_PASSES=0
      NUMNZ=N
      NUMNZ_NEW=0
      SWITCH=.FALSE.
      DO 91 I=1,N
        MARK(I)=.FALSE.
        SCAN(I)=.FALSE.
91    CONTINUE
      NLABEL=0
C
C     RELAX4 USES AN ADAPTIVE STRATEGY TO DECIDE WHETHER TO
C     CONTINUE THE SCANNING PROCESS AFTER A MULTINODE PRICE
CHANGE.
C     THE THRESHOLD PARAMETER TP AND TS THAT CONTROL
C     THIS STRATEGY ARE SET IN THE NEXT TWO LINES.
C
```

```
      TP=10
      TS=INT(N/15)
C
C     INITIALIZE THE QUEUE OF NODES WITH NONZERO DEFICIT
C
      DO 92 NODE=1,N-1
        NXTQUEUE(NODE)=NODE+1
92    CONTINUE
      NXTQUEUE(N)=1
      NODE=N
      LASTQUEUE=N
C
C--------------------------------------------------------------
C
C     START THE RELAXATION ALGORITHM.
C
100   CONTINUE
C
C     CODE FOR ADVANCING THE QUEUE OF NONZERO DEFICIT NODES
C
      PREVNODE=NODE
      NODE=NXTQUEUE(NODE)
      DEFCIT=DFCT(NODE)
      IF (NODE.EQ.LASTQUEUE) THEN
        NUMNZ=NUMNZ_NEW
        NUMNZ_NEW=0
        LASTQUEUE=PREVNODE
        NUM_PASSES=NUM_PASSES+1
      END IF
C
C     CODE FOR DELETING A NODE FROM THE QUEUE
C
      IF (DEFCIT.EQ.0) THEN
        NXTNODE=NXTQUEUE(NODE)
        IF (NODE.EQ.NXTNODE) THEN
          RETURN
        ELSE
          NXTQUEUE(PREVNODE)=NXTNODE
          NXTQUEUE(NODE)=0
          NODE=NXTNODE
          GO TO 100
        END IF
      ELSE
        POSIT = (DEFCIT.GT.0)
      END IF
C
```

```
      ITER=ITER+1
      NUMNZ_NEW=NUMNZ_NEW+1
C
      IF (POSIT) THEN
C
C    ATTEMPT A SINGLE NODE ITERATION FROM NODE WITH POSITIVE
DEFICIT
C
      PCHANGE = .FALSE.
      INDEF=DEFCIT
      DELX=0
      NB=0
C
C    CHECK OUTGOING (PROBABLY) BALANCED ARCS FROM NODE.
C
      ARC=TFSTOU(NODE)
4500  IF (ARC.GT.0) THEN
        IF ((RC(ARC).EQ.0).AND.(X(ARC).GT.0)) THEN
          DELX = DELX + X(ARC)
          NB = NB + 1
          SAVE(NB) = ARC
        ENDIF
        ARC = TNXTOU(ARC)
        GOTO 4500
      END IF
C
C    CHECK INCOMING ARCS.
C
      ARC = TFSTIN(NODE)
4501   IF (ARC.GT.0) THEN
         IF ((RC(ARC).EQ.0).AND.(U(ARC).GT.0)) THEN
           DELX = DELX + U(ARC)
           NB = NB + 1
           SAVE(NB) = -ARC
         ENDIF
         ARC = TNXTIN(ARC)
         GOTO 4501
      END IF
C
C    END OF INITIAL NODE SCAN.
C
4018    CONTINUE
C
C    IF NO PRICE CHANGE IS POSSIBLE, EXIT.
C
      IF (DELX.GT.DEFCIT) THEN
```

```
        QUIT = (DEFCIT .LT. INDEF)
        GO TO 4016
      END IF
C
C     RELAX4 SEARCHES ALONG THE ASCENT DIRECTION FOR THE
C     BEST PRICE BY CHECKING THE SLOPE OF THE DUAL COST
C     AT SUCCESSIVE BREAK POINTS.  FIRST, WE
C     COMPUTE THE DISTANCE TO THE NEXT BREAK POINT.
C
      DELPRC = LARGE
      ARC = FOU(NODE)
4502  IF (ARC .GT. 0) THEN
        RDCOST = RC(ARC)
        IF ((RDCOST.LT.0).AND.(RDCOST.GT.-DELPRC)) THEN
          DELPRC = -RDCOST
        END IF
        ARC = NXTOU(ARC)
        GOTO 4502
      END IF
      ARC = FIN(NODE)
4503  IF (ARC .GT. 0) THEN
        RDCOST = RC(ARC)
        IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) THEN
          DELPRC = RDCOST
        END IF
        ARC = NXTIN(ARC)
        GOTO 4503
      END IF
C
C     CHECK IF PROBLEM IS INFEASIBLE.
C
      IF ((DELX.LT.DEFCIT).AND.(DELPRC.EQ.LARGE)) THEN
C
C     THE DUAL COST CAN BE DECREASED WITHOUT BOUND.
C
        GO TO 4400
      END IF
C
C     SKIP FLOW ADJUSTEMT IF THERE IS NO FLOW TO MODIFY.
C
      IF (DELX.EQ.0) GO TO 4014
C
C     ADJUST THE FLOW ON THE BALANCED ARCS INCIDENT TO NODE TO
C     MAINTAIN COMPLEMENTARY SLACKNESS AFTER THE PRICE CHANGE.
C
      DO 4013 J=1,NB
```

```
       ARC=SAVE(J)
       IF (ARC.GT.0) THEN
        NODE2=ENDN(ARC)
        T1=X(ARC)
        DFCT(NODE2)=DFCT(NODE2)+T1
        IF (NXTQUEUE(NODE2).EQ.0) THEN
         NXTQUEUE(PREVNODE)=NODE2
         NXTQUEUE(NODE2)=NODE
         PREVNODE=NODE2
        END IF
        U(ARC)=U(ARC)+T1
        X(ARC)=0
       ELSE
        NARC=-ARC
        NODE2=STARTN(NARC)
        T1=U(NARC)
        DFCT(NODE2)=DFCT(NODE2)+T1
        IF (NXTQUEUE(NODE2).EQ.0) THEN
         NXTQUEUE(PREVNODE)=NODE2
         NXTQUEUE(NODE2)=NODE
         PREVNODE=NODE2
        END IF
        X(NARC)=X(NARC)+T1
        U(NARC)=0
       END IF
4013  CONTINUE
      DEFCIT=DEFCIT-DELX
4014  IF (DELPRC.EQ.LARGE) THEN
       QUIT=.TRUE.
       GO TO 4019
      END IF
C
C    NODE CORRESPONDS TO A DUAL ASCENT DIRECTION.  DECREASE
C    THE PRICE OF NODE BY DELPRC AND COMPUTE THE STEPSIZE TO THE
C    NEXT BREAKPOINT IN THE DUAL COST.
C
      NB=0
      PCHANGE = .TRUE.
      DP=DELPRC
      DELPRC=LARGE
      DELX=0
      ARC=FOU(NODE)
4504  IF (ARC.GT.0) THEN
       RDCOST=RC(ARC)+DP
       RC(ARC)=RDCOST
       IF (RDCOST.EQ.0) THEN
```

```
          NB=NB+1
          SAVE(NB)=ARC
          DELX=DELX+X(ARC)
         END IF
         IF ((RDCOST.LT.0).AND.(RDCOST.GT.-DELPRC)) DELPRC=-RDCOST
         ARC=NXTOU(ARC)
         GOTO 4504
        END IF
        ARC=FIN(NODE)
4505  IF (ARC.GT.0) THEN
         RDCOST=RC(ARC)-DP
         RC(ARC)=RDCOST
         IF (RDCOST.EQ.0) THEN
          NB=NB+1
          SAVE(NB)=-ARC
          DELX=DELX+U(ARC)
         END IF
         IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
         ARC=NXTIN(ARC)
         GOTO 4505
        END IF
C
C   RETURN TO CHECK IF ANOTHER PRICE CHANGE IS POSSIBLE.
C
        GO TO 4018
C
C   PERFORM FLOW AUGMENTATION AT NODE.
C
4016  DO 4011 J=1,NB
         ARC=SAVE(J)
         IF (ARC.GT.0) THEN
C
C   ARC IS AN OUTGOING ARC FROM NODE.
C
          NODE2=ENDN(ARC)
          T1=DFCT(NODE2)
          IF (T1.LT.0) THEN
C
C   DECREASE THE TOTAL DEFICIT BY DECREASING FLOW OF ARC.
C
           QUIT=.TRUE.
           T2=X(ARC)
           DX=MIN0(DEFCIT,-T1,T2)
           DEFCIT=DEFCIT-DX
           DFCT(NODE2)=T1+DX
           IF (NXTQUEUE(NODE2).EQ.0) THEN
```

```
            NXTQUEUE(PREVNODE)=NODE2
            NXTQUEUE(NODE2)=NODE
            PREVNODE=NODE2
          END IF
          X(ARC)=T2-DX
          U(ARC)=U(ARC)+DX
          IF (DEFCIT.EQ.0) GO TO 4019
        END IF
      ELSE
C
C    -ARC IS AN INCOMING ARC TO NODE.
C
        NARC=-ARC
        NODE2=STARTN(NARC)
        T1=DFCT(NODE2)
        IF (T1.LT.0) THEN
C
C    DECREASE THE TOTAL DEFICIT BY INCREASING FLOW OF -ARC.
C
          QUIT=.TRUE.
          T2=U(NARC)
          DX=MIN0(DEFCIT,-T1,T2)
          DEFCIT=DEFCIT-DX
          DFCT(NODE2)=T1+DX
          IF (NXTQUEUE(NODE2).EQ.0) THEN
            NXTQUEUE(PREVNODE)=NODE2
            NXTQUEUE(NODE2)=NODE
            PREVNODE=NODE2
          END IF
          X(NARC)=X(NARC)+DX
          U(NARC)=T2-DX
          IF (DEFCIT.EQ.0) GO TO 4019
        END IF
      END IF
4011  CONTINUE
4019  DFCT(NODE)=DEFCIT
C
C    RECONSTRUCT THE LINKED LIST OF BALANCE ARCS INCIDENT TO
THIS NODE.
C     FOR EACH ADJACENT NODE, WE ADD ANY NEWLY BLANCED ARCS
C     TO THE LIST, BUT DO NOT BOTHER REMOVING FORMERLY BALANCED
ONES
C     (THEY WILL BE REMOVED THE NEXT TIME EACH ADJACENT NODE IS
SCANNED).
C
    IF (PCHANGE) THEN
```

```
      ARC = TFSTOU(NODE)
      TFSTOU(NODE) = 0
4506   IF (ARC .GT. 0) THEN
        NXTARC = TNXTOU(ARC)
        TNXTOU(ARC) = -1
        ARC = NXTARC
        GOTO 4506
      END IF
      ARC = TFSTIN(NODE)
      TFSTIN(NODE) = 0
4507   IF (ARC .GT. 0) THEN
        NXTARC = TNXTIN(ARC)
        TNXTIN(ARC) = -1
        ARC = NXTARC
        GOTO 4507
        END IF
C
C   NOW ADD THE CURRENTLY BALANCED ARCS TO THE LIST FOR THIS
NODE
C   (WHICH IS NOW EMPTY), AND THE APPROPRIATE ADJACENT ONES.
C
        DO 4508 J=1,NB
         ARC = SAVE(J)
         IF (ARC.LE.0) ARC=-ARC
         IF (TNXTOU(ARC) .LT. 0) THEN
          TNXTOU(ARC) = TFSTOU(STARTN(ARC))
          TFSTOU(STARTN(ARC)) = ARC
         END IF
         IF (TNXTIN(ARC) .LT. 0) THEN
          TNXTIN(ARC) = TFSTIN(ENDN(ARC))
          TFSTIN(ENDN(ARC)) = ARC
         END IF
4508     CONTINUE

     END IF
C
C   END OF SINGLE NODE ITERATION FOR POSITIVE DEFICIT NODE.
C
    ELSE
C
C   ATTEMPT A SINGLE NODE ITERATION FROM NODE WITH NEGATIVE
DEFICIT
C
    PCHANGE = .FALSE.
    DEFCIT=-DEFCIT
    INDEF=DEFCIT
```

```
      DELX=0
      NB=0
C
      ARC = TFSTIN(NODE)
 4509 IF (ARC .GT. 0) THEN
        IF ((RC(ARC) .EQ. 0) .AND. (X(ARC) .GT. 0)) THEN
          DELX = DELX + X(ARC)
          NB = NB + 1
          SAVE(NB) = ARC
        END IF
        ARC = TNXTIN(ARC)
        GOTO 4509
      END IF
      ARC=TFSTOU(NODE)
 4510 IF (ARC .GT. 0) THEN
        IF ((RC(ARC) .EQ. 0) .AND. (U(ARC) .GT. 0)) THEN
          DELX = DELX + U(ARC)
          NB = NB + 1
          SAVE(NB) = -ARC
        END IF
        ARC = TNXTOU(ARC)
        GOTO 4510
      END IF
C
 4028 CONTINUE
      IF (DELX.GE.DEFCIT) THEN
        QUIT = (DEFCIT .LT. INDEF)
        GO TO 4026
      END IF
C
C     COMPUTE DISTANCE TO NEXT BREAKPOINT.
C
      DELPRC = LARGE
      ARC = FIN(NODE)
 4511 IF (ARC .GT. 0) THEN
        RDCOST = RC(ARC)
        IF ((RDCOST .LT. 0) .AND. (RDCOST.GT.-DELPRC)) THEN
          DELPRC = -RDCOST
        END IF
        ARC = NXTIN(ARC)
        GOTO 4511
      END IF
      ARC = FOU(NODE)
 4512 IF (ARC .GT. 0) THEN
        RDCOST = RC(ARC)
        IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) THEN
        DELPRC = RDCOST
       END IF
       ARC = NXTOU(ARC)
       GOTO 4512
      END IF
C
C   CHECK IF PROBLEM IS INFEASIBLE.
C
      IF ((DELX.LT.DEFCIT).AND.(DELPRC.EQ.LARGE)) THEN
       GO TO 4400
      END IF
      IF (DELX.EQ.0) GO TO 4024
C
C   FLOW AUGMENTATION IS POSSIBLE.
C
      DO 4023 J=1,NB
       ARC=SAVE(J)
       IF (ARC.GT.0) THEN
        NODE2=STARTN(ARC)
        T1=X(ARC)
        DFCT(NODE2)=DFCT(NODE2)-T1
        IF (NXTQUEUE(NODE2).EQ.0) THEN
         NXTQUEUE(PREVNODE)=NODE2
         NXTQUEUE(NODE2)=NODE
         PREVNODE=NODE2
        END IF
        U(ARC)=U(ARC)+T1
        X(ARC)=0
       ELSE
        NARC=-ARC
        NODE2=ENDN(NARC)
        T1=U(NARC)
        DFCT(NODE2)=DFCT(NODE2)-T1
        IF (NXTQUEUE(NODE2).EQ.0) THEN
         NXTQUEUE(PREVNODE)=NODE2
         NXTQUEUE(NODE2)=NODE
         PREVNODE=NODE2
        END IF
        X(NARC)=X(NARC)+T1
        U(NARC)=0
       END IF
4023  CONTINUE
      DEFCIT=DEFCIT-DELX
4024  IF (DELPRC.EQ.LARGE) THEN
       QUIT=.TRUE.
       GO TO 4029
      END IF
C
C     PRICE INCREASE AT NODE IS POSSIBLE.
C
      NB=0
      PCHANGE = .TRUE.
      DP=DELPRC
      DELPRC=LARGE
      DELX=0
      ARC=FIN(NODE)
4513  IF (ARC.GT.0) THEN
        RDCOST=RC(ARC)+DP
        RC(ARC)=RDCOST
        IF (RDCOST.EQ.0) THEN
          NB=NB+1
          SAVE(NB)=ARC
          DELX=DELX+X(ARC)
        END IF
        IF ((RDCOST.LT.0).AND.(RDCOST.GT.-DELPRC)) DELPRC=-RDCOST
        ARC=NXTIN(ARC)
        GOTO 4513
      END IF
      ARC=FOU(NODE)
4514  IF (ARC.GT.0) THEN
        RDCOST=RC(ARC)-DP
        RC(ARC)=RDCOST
        IF (RDCOST.EQ.0) THEN
          NB=NB+1
          SAVE(NB)=-ARC
          DELX=DELX+U(ARC)
        END IF
        IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
        ARC=NXTOU(ARC)
        GOTO 4514
      END IF
      GO TO 4028
C
C     PERFORM FLOW AUGMENTATION AT NODE.
C
4026    DO 4021 J=1,NB
        ARC=SAVE(J)
        IF (ARC.GT.0) THEN
C
C     ARC IS AN INCOMING ARC TO NODE.
C
          NODE2=STARTN(ARC)
```

```
         T1=DFCT(NODE2)
         IF (T1.GT.0) THEN
          QUIT=.TRUE.
          T2=X(ARC)
          DX=MIN0(DEFCIT,T1,T2)
          DEFCIT=DEFCIT-DX
          DFCT(NODE2)=T1-DX
          IF (NXTQUEUE(NODE2).EQ.0) THEN
           NXTQUEUE(PREVNODE)=NODE2
           NXTQUEUE(NODE2)=NODE
           PREVNODE=NODE2
          END IF
          X(ARC)=T2-DX
          U(ARC)=U(ARC)+DX
          IF (DEFCIT.EQ.0) GO TO 4029
         END IF
        ELSE
C
C    -ARC IS AN OUTGOING ARC FROM NODE.
C
         NARC=-ARC
         NODE2=ENDN(NARC)
         T1=DFCT(NODE2)
         IF (T1.GT.0) THEN
          QUIT=.TRUE.
          T2=U(NARC)
          DX=MIN0(DEFCIT,T1,T2)
          DEFCIT=DEFCIT-DX
          DFCT(NODE2)=T1-DX
          IF (NXTQUEUE(NODE2).EQ.0) THEN
           NXTQUEUE(PREVNODE)=NODE2
           NXTQUEUE(NODE2)=NODE
           PREVNODE=NODE2
          END IF
          X(NARC)=X(NARC)+DX
          U(NARC)=T2-DX
          IF (DEFCIT.EQ.0) GO TO 4029
         END IF
        END IF
4021  CONTINUE
4029  DFCT(NODE)=-DEFCIT
C
C    RECONSTRUCT THE LIST OF BALANCED ARCS INCIDENT TO NODE.
C
      IF (PCHANGE) THEN
      ARC = TFSTOU(NODE)
      TFSTOU(NODE) = 0
 4515   IF (ARC .GT. 0) THEN
         NXTARC = TNXTOU(ARC)
         TNXTOU(ARC) = -1
         ARC = NXTARC
         GOTO 4515
       END IF
       ARC = TFSTIN(NODE)
       TFSTIN(NODE) = 0
 4516   IF (ARC .GT. 0) THEN
         NXTARC = TNXTIN(ARC)
         TNXTIN(ARC) = -1
         ARC = NXTARC
         GOTO 4516
       END IF
C
C   NOW ADD THE CURRENTLY BALANCED ARCS TO THE LIST FOR THIS
NODE
C   (WHICH IS NOW EMPTY), AND THE APPROPRIATE ADJACENT ONES.
C
       DO 4517 J=1,NB
        ARC = SAVE(J)
        IF (ARC.LE.0) ARC=-ARC
        IF (TNXTOU(ARC) .LT. 0) THEN
         TNXTOU(ARC) = TFSTOU(STARTN(ARC))
         TFSTOU(STARTN(ARC)) = ARC
        END IF
        IF (TNXTIN(ARC) .LT. 0) THEN
         TNXTIN(ARC) = TFSTIN(ENDN(ARC))
         TFSTIN(ENDN(ARC)) = ARC
        END IF
4517   CONTINUE

     END IF
C
C    END OF SINGLE NODE ITERATION FOR A NEGATIVE DEFICIT NODE.
C
     END IF
C
     IF (QUIT.OR.(NUM_PASSES.LE.3)) GO TO 100
C
C    DO A MULTINODE ITERATION FROM NODE.
C
     NMULTINODE=NMULTINODE+1
C
```

```
C    IF NUMBER OF NONZERO DEFICIT NODES IS SMALL, CONTINUE
C    LABELING UNTIL A FLOW AUGMENTATION IS DONE.
C
      SWITCH = (NUMNZ.LT.TP)
C
C    UNMARK NODES LABELED EARLIER.
C
      DO 4090 J=1,NLABEL
       NODE2=LABEL(J)
       MARK(NODE2)=.FALSE.
       SCAN(NODE2)=.FALSE.
4090  CONTINUE
C
C    INITIALIZE LABELING.
C
      NLABEL=1
      LABEL(1)=NODE
      MARK(NODE)=.TRUE.
      PRDCSR(NODE)=0
C
C    SCAN STARTING NODE.
C
      SCAN(NODE)=.TRUE.
      NSCAN=1
      DM=DFCT(NODE)
      DELX=0
      DO 4095 J=1,NB
       ARC=SAVE(J)
       IF (ARC.GT.0) THEN
        IF (POSIT) THEN
         NODE2=ENDN(ARC)
        ELSE
         NODE2=STARTN(ARC)
        END IF
        IF (.NOT.MARK(NODE2)) THEN
         NLABEL=NLABEL+1
         LABEL(NLABEL)=NODE2
         PRDCSR(NODE2)=ARC
         MARK(NODE2)=.TRUE.
         DELX=DELX+X(ARC)
        END IF
       ELSE
        NARC=-ARC
        IF (POSIT) THEN
         NODE2=STARTN(NARC)
        ELSE
```

```
          NODE2=ENDN(NARC)
         END IF
         IF (.NOT.MARK(NODE2)) THEN
          NLABEL=NLABEL+1
          LABEL(NLABEL)=NODE2
          PRDCSR(NODE2)=ARC
          MARK(NODE2)=.TRUE.
          DELX=DELX+U(NARC)
         END IF
        END IF
4095  CONTINUE
C
C    START SCANNING A LABELED BUT UNSCANNED NODE.
C
4120  NSCAN=NSCAN+1
C
C    CHECK TO SEE IF SWITCH NEEDS TO BE SET TO TRUE SO TO
C    CONTINUE SCANNING EVEN AFTER A PRICE CHANGE.
C
      SWITCH = SWITCH .OR.
     $((NSCAN .GT. TS).AND.(NUMNZ.LT.TS))
C
C    SCANNING WILL CONTINUE UNTIL EITHER AN OVERESTIMATE OF THE
RESIDUAL
C    CAPACITY ACROSS THE CUT CORRESPONDING TO THE SCANNED SET
OF NODES (CALLED
C    DELX) EXCEEDS THE ABSOLUTE VALUE OF THE TOTAL DEFICIT OF
THE SCANNED
C    NODES (CALLED DM), OR ELSE AN AUGMENTING PATH IS FOUND.
ARCS THAT ARE
C    IN THE TREE BUT ARE NOT BALANCED ARE REMOVED AS PART OF THE
SCANNING
C    PROCESS.
C
      I=LABEL(NSCAN)
      SCAN(I)=.TRUE.
      NAUGNOD=0
      IF (POSIT) THEN
C
C    SCANNING NODE I IN CASE OF POSITIVE DEFICIT.
C
        PRVARC=0
        ARC = TFSTOU(I)

4518   IF (ARC.GT.0) THEN
C
```

```
C     ARC IS AN OUTGOING ARC FROM NODE.
C
       IF (RC(ARC) .EQ. 0) THEN
        IF (X(ARC) .GT. 0) THEN
         NODE2=ENDN(ARC)
         IF (.NOT. MARK(NODE2)) THEN
C
C     NODE2 IS NOT LABELED, SO ADD NODE2 TO THE LABELED SET.
C
          PRDCSR(NODE2)=ARC
          IF (DFCT(NODE2).LT.0) THEN
           NAUGNOD=NAUGNOD+1
           SAVE(NAUGNOD)=NODE2
          END IF
          NLABEL=NLABEL+1
          LABEL(NLABEL)=NODE2
          MARK(NODE2)=.TRUE.
          DELX=DELX+X(ARC)
         END IF
        END IF
        PRVARC = ARC
        ARC = TNXTOU(ARC)
       ELSE
        TMPARC = ARC
        ARC = TNXTOU(ARC)
        TNXTOU(TMPARC) = -1
        IF (PRVARC .EQ. 0) THEN
         TFSTOU(I) = ARC
        ELSE
         TNXTOU(PRVARC) = ARC
        END IF
       END IF
       GOTO 4518
      END IF

      PRVARC = 0
      ARC=TFSTIN(I)
4519    IF (ARC.GT.0) THEN
C
C     ARC IS AN INCOMING ARC INTO NODE.
C
       IF (RC(ARC) .EQ. 0) THEN
        IF (U(ARC) .GT. 0) THEN
         NODE2=STARTN(ARC)
         IF (.NOT. MARK(NODE2)) THEN
C
```

```
C   NODE2 IS NOT LABELED, SO ADD NODE2 TO THE LABELED SET.
C
           PRDCSR(NODE2)=-ARC
           IF (DFCT(NODE2).LT.0) THEN
            NAUGNOD=NAUGNOD+1
            SAVE(NAUGNOD)=NODE2
           END IF
           NLABEL=NLABEL+1
           LABEL(NLABEL)=NODE2
           MARK(NODE2)=.TRUE.
           DELX=DELX+U(ARC)
          END IF
         END IF
         PRVARC = ARC
         ARC = TNXTIN(ARC)
        ELSE
         TMPARC = ARC
         ARC = TNXTIN(ARC)
         TNXTIN(TMPARC) = -1
         IF (PRVARC .EQ. 0) THEN
          TFSTIN(I) = ARC
         ELSE
          TNXTIN(PRVARC) = ARC
         END IF
        END IF
        GOTO 4519
       END IF
C
C   CORRECT THE RESIDUAL CAPACITY OF THE SCANNED NODE CUT.
C
       ARC=PRDCSR(I)
       IF (ARC.GT.0) THEN
        DELX=DELX-X(ARC)
       ELSE
        DELX=DELX-U(-ARC)
       END IF
C
C   END OF SCANNING OF NODE I FOR POSITIVE DEFICIT CASE.
C
      ELSE
C
C   SCANNING NODE I FOR NEGATIVE DEFICIT CASE.
C
       PRVARC = 0
       ARC=TFSTIN(I)
4520    IF (ARC.GT.0) THEN
```

```
        IF (RC(ARC) .EQ. 0) THEN
         IF (X(ARC) .GT. 0) THEN
          NODE2=STARTN(ARC)
          IF (.NOT. MARK(NODE2)) THEN
           PRDCSR(NODE2)=ARC
           IF (DFCT(NODE2).GT.0) THEN
            NAUGNOD=NAUGNOD+1
            SAVE(NAUGNOD)=NODE2
           END IF
           NLABEL=NLABEL+1
           LABEL(NLABEL)=NODE2
           MARK(NODE2)=.TRUE.
           DELX=DELX+X(ARC)
          END IF
         END IF
         PRVARC = ARC
         ARC = TNXTIN(ARC)
        ELSE
         TMPARC = ARC
         ARC = TNXTIN(ARC)
         TNXTIN(TMPARC) = -1
         IF (PRVARC .EQ. 0) THEN
          TFSTIN(I) = ARC
         ELSE
          TNXTIN(PRVARC) = ARC
         END IF
        END IF
       GOTO 4520
      END IF
C
      PRVARC = 0
      ARC = TFSTOU(I)
4521  IF (ARC.GT.0) THEN
        IF (RC(ARC) .EQ. 0) THEN
         IF (U(ARC) .GT. 0) THEN
          NODE2=ENDN(ARC)
          IF (.NOT. MARK(NODE2)) THEN
           PRDCSR(NODE2)=-ARC
           IF (DFCT(NODE2).GT.0) THEN
            NAUGNOD=NAUGNOD+1
            SAVE(NAUGNOD)=NODE2
           END IF
           NLABEL=NLABEL+1
           LABEL(NLABEL)=NODE2
           MARK(NODE2)=.TRUE.
           DELX=DELX+U(ARC)
          END IF
         END IF
         PRVARC = ARC
         ARC = TNXTOU(ARC)
        ELSE
         TMPARC = ARC
         ARC = TNXTOU(ARC)
         TNXTOU(TMPARC) = -1
         IF (PRVARC .EQ. 0) THEN
          TFSTOU(I) = ARC
         ELSE
          TNXTOU(PRVARC) = ARC
         END IF
        END IF
        GOTO 4521
       END IF
C
       ARC=PRDCSR(I)
       IF (ARC.GT.0) THEN
        DELX=DELX-X(ARC)
       ELSE
        DELX=DELX-U(-ARC)
       END IF
      END IF
C
C   ADD DEFICIT OF NODE SCANNED TO DM.
C
      DM=DM+DFCT(I)
C
C   CHECK IF THE SET OF SCANNED NODES CORRESPOND
C   TO A DUAL ASCENT DIRECTION; IF YES, PERFORM A
C   PRICE ADJUSTMENT STEP, OTHERWISE CONTINUE LABELING.
C
      IF (NSCAN.LT.NLABEL) THEN
       IF (SWITCH) GO TO 4210
       IF ((DELX.GE.DM).AND.(DELX.GE.-DM)) GO TO 4210
      END IF
C
C   TRY A PRICE CHANGE.
C   [NOTE THAT SINCE DELX-ABS(DM) IS AN OVERESTIMATE OF ASCENT
SLOPE, WE
C   MAY OCCASIONALLY TRY A DIRECTION THAT IS NOT AN ASCENT
DIRECTION.
C   IN THIS CASE, THE ASCNT ROUTINES RETURN WITH QUIT=.FALSE.,
C   SO WE CONTINUE LABELING NODES.
C
```

```
      IF (POSIT) THEN
       CALL ASCNT1(DM,DELX,NLABEL,FEASBL,
     $  SWITCH,NSCAN,NODE,PREVNODE)
       NUM_ASCNT=NUM_ASCNT+1
      ELSE
       CALL ASCNT2(DM,DELX,NLABEL,FEASBL,
     $  SWITCH,NSCAN,NODE,PREVNODE)
       NUM_ASCNT=NUM_ASCNT+1
      END IF
      IF (.NOT.FEASBL) GO TO 4400
      IF (.NOT.SWITCH) GO TO 100
C
C    STORE THOSE NEWLY LABELED NODES TO WHICH FLOW
AUGMENTATION IS POSSIBLE.
C
      NAUGNOD=0
      DO 530 J=NSCAN+1,NLABEL
       NODE2=LABEL(J)
       IF (POSIT.AND.(DFCT(NODE2).LT.0)) THEN
        NAUGNOD=NAUGNOD+1
        SAVE(NAUGNOD)=NODE2
       ELSE IF ((.NOT.POSIT).AND.(DFCT(NODE2).GT.0)) THEN
        NAUGNOD=NAUGNOD+1
        SAVE(NAUGNOD)=NODE2
       END IF
530   CONTINUE
C
C    CHECK IF FLOW AUGMENTATION IS POSSIBLE.
C    IF NOT, RETURN TO SCAN ANOTHER NODE.
C
4210  CONTINUE
C
      IF (NAUGNOD.EQ.0) GO TO 4120
C
      DO 4096 J=1,NAUGNOD
       NUM_AUGM=NUM_AUGM+1
       AUGNOD=SAVE(J)
       IF (POSIT) THEN
C
C    DO THE AUGMENTATION FROM NODE WITH POSITIVE DEFICIT.
C
        DX=-DFCT(AUGNOD)
        IB=AUGNOD
1500     IF (IB.NE.NODE) THEN
         ARC=PRDCSR(IB)
         IF (ARC.GT.0) THEN
        DX=MIN0(DX,X(ARC))
        IB=STARTN(ARC)
       ELSE
        DX=MIN0(DX,U(-ARC))
        IB=ENDN(-ARC)
       END IF
       GOTO 1500
      END IF
      DX=MIN0(DX,DFCT(NODE))
      IF (DX .GT. 0) THEN
C
C    INCREASE (DECREASE) THE FLOW OF ALL FORWARD (BACKWARD)
C    ARCS IN THE FLOW AUGMENTING PATH.  ADJUST NODE DEFICIT
ACCORDINGLY.
C
       IF (NXTQUEUE(AUGNOD).EQ.0) THEN
        NXTQUEUE(PREVNODE)=AUGNOD
        NXTQUEUE(AUGNOD)=NODE
        PREVNODE=AUGNOD
       END IF
       DFCT(AUGNOD)=DFCT(AUGNOD)+DX
       DFCT(NODE)=DFCT(NODE)-DX
       IB=AUGNOD
1501     IF (IB.NE.NODE) THEN
        ARC=PRDCSR(IB)
        IF (ARC.GT.0) THEN
         X(ARC)=X(ARC)-DX
         U(ARC)=U(ARC)+DX
         IB=STARTN(ARC)
        ELSE
         NARC=-ARC
         X(NARC)=X(NARC)+DX
         U(NARC)=U(NARC)-DX
         IB=ENDN(NARC)
        END IF
        GOTO 1501
       END IF
      END IF
     ELSE
C
C    DO THE AUGMENTATION FROM NODE WITH NEGATIVE DEFICIT.
C
      DX=DFCT(AUGNOD)
      IB=AUGNOD
1502    IF (IB.NE.NODE) THEN
       ARC=PRDCSR(IB)
      IF (ARC.GT.0) THEN
       DX=MIN0(DX,X(ARC))
       IB=ENDN(ARC)
      ELSE
       DX=MIN0(DX,U(-ARC))
       IB=STARTN(-ARC)
      END IF
      GOTO 1502
     END IF
     DX=MIN0(DX,-DFCT(NODE))
     IF (DX .GT. 0) THEN
C
C   UPDATE THE FLOW AND DEFICITS.
C
      IF (NXTQUEUE(AUGNOD).EQ.0) THEN
       NXTQUEUE(PREVNODE)=AUGNOD
       NXTQUEUE(AUGNOD)=NODE
       PREVNODE=AUGNOD
      END IF
      DFCT(AUGNOD)=DFCT(AUGNOD)-DX
      DFCT(NODE)=DFCT(NODE)+DX
      IB=AUGNOD
1503    IF (IB.NE.NODE) THEN
       ARC=PRDCSR(IB)
       IF (ARC.GT.0) THEN
        X(ARC)=X(ARC)-DX
        U(ARC)=U(ARC)+DX
        IB=ENDN(ARC)
       ELSE
        NARC=-ARC
        X(NARC)=X(NARC)+DX
        U(NARC)=U(NARC)-DX
        IB=STARTN(NARC)
       END IF
       GOTO 1503
      END IF
     END IF
    END IF
    IF (DFCT(NODE).EQ.0) GO TO 100
    IF (DFCT(AUGNOD).NE.0) SWITCH=.FALSE.
4096 CONTINUE
C
C   IF NODE STILL HAS NONZERO DEFICIT AND ALL NEWLY
C   LABELED NODES HAVE SAME SIGN FOR THEIR DEFICIT AS
C   NODE, WE CAN CONTINUE LABELING.  IN THIS CASE, CONTINUE
C   LABELING ONLY WHEN FLOW AUGMENTATION IS DONE
```

```
C   RELATIVELY INFREQUENTLY.
C
      IF (SWITCH.AND.(ITER.GT.8*NUM_AUGM)) GO TO 4120
C
C   RETURN TO DO ANOTHER RELAXATION ITERATION.
C
      GO TO 100
C
C   PROBLEM IS FOUND TO BE INFEASIBLE
C
4400  PRINT*,' PROBLEM IS FOUND TO BE INFEASIBLE.'
      PRINT*, 'PROGRAM ENDED; PRESS <CR> TO EXIT'
      PAUSE
      STOP
C
      END
C
C
      SUBROUTINE AUCTION
      IMPLICIT INTEGER (A-Z)
C
C-----------------------------------------------------------------
C
C  PURPOSE - THIS SUBROUTINE USES A VERSION OF THE AUCTION
C    ALGORITHM FOR MIN COST NETWORK FLOW TO COMPUTE A
C    GOOD INITIAL FLOW AND PRICES FOR THE PROBLEM.
C
C-----------------------------------------------------------------
C
C    MAXNN = DIMENSION OF NODE-LENGTH ARRAYS
C    MAXNA = DIMENSION OF ARC-LENGTH ARRAYS
C
      PARAMETER (MAXNN=10000, MAXNA=70000)
C
C  INPUT PARAMETERS
C
C    N         = NUMBER OF NODES
C    NA        = NUMBER OF ARCS
C    LARGE     = A VERY LARGE INTEGER TO REPRESENT INFINITY
C                (SEE NOTE 3)
C    STARTN(I) = STARTING NODE FOR THE I-TH ARC,    I = 1,...,NA
C    ENDN(I)   = ENDING NODE FOR THE I-TH ARC,      I = 1,...,NA
C    FOU(I)    = FIRST ARC LEAVING I-TH NODE,       I = 1,...,N
C    NXTOU(I)  = NEXT ARC LEAVING THE STARTING NODE OF J-TH ARC,
C                                   I = 1,...,NA
C    FIN(I)    = FIRST ARC ENTERING I-TH NODE,      I = 1,...,N
```

```
C    NXTIN(I)  = NEXT ARC ENTERING THE ENDING NODE OF J-TH ARC,
C                                       I = 1,...,NA
C
      INTEGER STARTN(MAXNA),ENDN(MAXNA)
      INTEGER FOU(MAXNN),NXTOU(MAXNA),FIN(MAXNN),NXTIN(MAXNA)
      COMMON /INPUT/N,NA,LARGE
      COMMON /ARRAYS/STARTN/ARRAYE/ENDN
      COMMON /BLK3/FOU/BLK4/NXTOU/BLK5/FIN/BLK6/NXTIN
      COMMON /CR/CRASH
C
C  UPDATED PARAMETERS
C
C    RC(J)     = REDUCED COST OF ARC J,            J = 1,...,NA
C    U(J)      = RESIDUAL CAPACITY OF ARC J,
C                                       J = 1,...,NA
C    X(J)      = FLOW ON ARC J,                  J = 1,...,NA
C    DFCT(I)   = DEFICIT AT NODE I,              I = 1,...,N
C
      INTEGER RC(MAXNA),U(MAXNA),X(MAXNA),DFCT(MAXNN)
      COMMON /ARRAYRC/RC/ARRAYU/U/ARRAYX/X/ARRAYB/DFCT
C
C  OUTPUT PARAMETERS
C
      COMMON /OUTPUT/NMULTINODE,ITER,NUM_AUGM,NUM_ASCNT,NSP
C
C  WORKING PARAMETERS
C
      INTEGER P(MAXNN),PRDCSR(MAXNN),SAVE(MAXNA)
      INTEGER FPUSHF(MAXNN),NXTPUSHF(MAXNA)
      INTEGER FPUSHB(MAXNN),NXTPUSHB(MAXNA)
      INTEGER NXTQUEUE(MAXNN),EXTEND_ARC(MAXNN)
      INTEGER SB_LEVEL(MAXNN),SB_ARC(MAXNN)
      LOGICAL*1 PATH_ID(MAXNN)
      COMMON /BLK1/P/BLK2/PRDCSR/BLK7/SAVE
      COMMON
/BLK10/FPUSHF/BLK11/NXTPUSHF/BLK12/FPUSHB/BLK13/NXTPUSHB
      COMMON /BLK14/NXTQUEUE/BLK15/EXTEND_ARC
      COMMON /BLK16/SB_LEVEL/BLK17/SB_ARC
      COMMON /BLK9/PATH_ID
C
C    START INITIALIZATION USING AUCTION
C
      NAUG=0
      PASS=0
      THRESH_DFCT=0
C
```

```
C   FACTOR DETERMINES BY HOW MUCH EPSILON IS REDUCED AT EACH
MINIMIZATION
C
    FACTOR=3
C
C   NUM_PASSES DETERMINES HOW MANY AUCTION SCALING PHASES
ARE PERFORMED
C
    NUM_PASSES=1

C   SET ARC FLOWS TO SATISFY CS AND CALCULATE MAXCOST AND
MINCOST

    MAXCOST=-LARGE
    MINCOST=LARGE
    DO 49 ARC=1,NA
     START=STARTN(ARC)
     END=ENDN(ARC)
     RDCOST=RC(ARC)
     IF (MAXCOST.LT.RDCOST) MAXCOST=RDCOST
     IF (MINCOST.GT.RDCOST) MINCOST=RDCOST
     IF (RDCOST.LT.0) THEN
      DFCT(START)=DFCT(START)+U(ARC)
      DFCT(END)=DFCT(END)-U(ARC)
      X(ARC)=U(ARC)
      U(ARC)=0
     ELSE
      X(ARC)=0
     END IF
49   CONTINUE
C
C   SET INITIAL EPSILON
C
    IF ((MAXCOST-MINCOST).GE.8) THEN
     EPS=INT((MAXCOST-MINCOST)/8)
    ELSE
     EPS=1
    END IF
C
C   SET INITIAL PRICES TO ZERO
C
    DO 48 NODE=1,N
     P(NODE)=0
48   CONTINUE
C
C   INITIALIZATION USING AUCTION/SHORTEST PATHS.
```

```
C   START OF THE FIRST SCALING PHASE.
C
100  CONTINUE

     PASS=PASS+1
     IF ((PASS.EQ.NUM_PASSES).OR.(EPS.EQ.1)) CRASH=0
     NOLIST=0
C
C   CONSTRUCT LIST OF POSITIVE SURPLUS NODES AND QUEUE OF
NEGATIVE SURPLUS
C   NODES
C
     DO 110 NODE=1,N
      PRDCSR(NODE)=0
      PATH_ID(NODE)=.FALSE.
      EXTEND_ARC(NODE)=0
      SB_LEVEL(NODE)=-LARGE
      NXTQUEUE(NODE)=NODE+1
      IF (DFCT(NODE).GT.0) THEN
       NOLIST=NOLIST+1
       SAVE(NOLIST)=NODE
      END IF
110  CONTINUE
C
     NXTQUEUE(N)=1
     ROOT=1
     PREVNODE=N
     LASTQUEUE=N
C
C   INITIALIZATION WITH DOWN ITERATIONS FOR NEGATIVE SURPLUS
NODES
C
     DO 150 I=1,NOLIST
      NODE=SAVE(I)
      NSP=NSP+1
C
C   BUILD THE LIST OF ARCS W/ ROOM FOR PUSHING FLOW
C   AND FIND PROPER PRICE FOR DOWN ITERATION
C
      BSTLEVEL=-LARGE
      FPUSHF(NODE)=0
      ARC=FOU(NODE)
152    IF (ARC.GT.0) THEN
       IF (U(ARC).GT.0) THEN
        IF (FPUSHF(NODE).EQ.0) THEN
         FPUSHF(NODE)=ARC
```

```
          NXTPUSHF(ARC)=0
          LAST=ARC
         ELSE
          NXTPUSHF(LAST)=ARC
          NXTPUSHF(ARC)=0
          LAST=ARC
         END IF
        END IF
        IF (X(ARC).GT.0) THEN
         NEW_LEVEL = P(ENDN(ARC)) + RC(ARC)
         IF (NEW_LEVEL.GT.BSTLEVEL) THEN
          BSTLEVEL=NEW_LEVEL
          EXTARC=ARC
         END IF
        END IF
        ARC=NXTOU(ARC)
        GO TO 152
       END IF
C
       FPUSHB(NODE)=0
       ARC=FIN(NODE)
154     IF (ARC.GT.0) THEN
        IF (X(ARC).GT.0) THEN
         IF (FPUSHB(NODE).EQ.0) THEN
          FPUSHB(NODE)=ARC
          NXTPUSHB(ARC)=0
          LAST=ARC
         ELSE
          NXTPUSHB(LAST)=ARC
          NXTPUSHB(ARC)=0
          LAST=ARC
         END IF
        END IF
        IF (U(ARC).GT.0) THEN
         NEW_LEVEL = P(STARTN(ARC)) - RC(ARC)
         IF (NEW_LEVEL.GT.BSTLEVEL) THEN
          BSTLEVEL=NEW_LEVEL
          EXTARC=-ARC
         END IF
        END IF
        ARC=NXTIN(ARC)
        GO TO 154
       END IF
       EXTEND_ARC(NODE)=EXTARC
       P(NODE)=BSTLEVEL-EPS
```

```
150  CONTINUE
C
C   START THE AUGMENTATION CYCLES OF THE NEW SCALING PHASE.
C
200  CONTINUE

   IF (DFCT(ROOT).GE.THRESH_DFCT) GOTO 3000

   TERM=ROOT
   PATH_ID(ROOT)=.TRUE.
C
C   MAIN FORWARD ALGORITHM WITH ROOT AS ORIGIN.
C
500  CONTINUE
C   START OF A NEW FORWARD ITERATION
C
   PTERM=P(TERM)
   EXTARC=EXTEND_ARC(TERM)

   IF (EXTARC.EQ.0) THEN
C
C   BUILD THE LIST OF ARCS W/ ROOM FOR PUSHING FLOW
C
    FPUSHF(TERM)=0
    ARC=FOU(TERM)
502    IF (ARC.GT.0) THEN
     IF (U(ARC).GT.0) THEN
      IF (FPUSHF(TERM).EQ.0) THEN
       FPUSHF(TERM)=ARC
       NXTPUSHF(ARC)=0
       LAST=ARC
      ELSE
       NXTPUSHF(LAST)=ARC
       NXTPUSHF(ARC)=0
       LAST=ARC
      END IF
     END IF
     ARC=NXTOU(ARC)
     GO TO 502
    END IF
C
    FPUSHB(TERM)=0
    ARC=FIN(TERM)
504    IF (ARC.GT.0) THEN
     IF (X(ARC).GT.0) THEN
      IF (FPUSHB(TERM).EQ.0) THEN
```

```
       FPUSHB(TERM)=ARC
       NXTPUSHB(ARC)=0
       LAST=ARC
      ELSE
       NXTPUSHB(LAST)=ARC
       NXTPUSHB(ARC)=0
       LAST=ARC
      END IF
     END IF
     ARC=NXTIN(ARC)
     GO TO 504
    END IF

    GO TO 600
   END IF
C
C   SPECULATIVE PATH EXTENSION ATTEMPT
C   NOTE: ARC>0 MEANS THAT ARC IS ORIENTED FROM THE ROOT TO THE
DESTINATIONS
C   ARC<0 MEANS THAT ARC IS ORIENTED FROM THE DESTINATIONS TO
THE ROOT
C   EXTARC=0 OR PRDARC=0, MEANS THE EXTENSION ARC OR THE
PREDECESSOR ARC,
C   RESPECTIVELY, HAS NOT BEEN ESTABLISHED
C
510  CONTINUE

   IF (EXTARC.GT.0) THEN

    IF (U(EXTARC).EQ.0) THEN
     SECLEVEL=SB_LEVEL(TERM)
     GO TO 580
    END IF
    END=ENDN(EXTARC)
    BSTLEVEL=P(END)+RC(EXTARC)
    IF (PTERM.GE.BSTLEVEL) THEN
     IF (PATH_ID(END)) GOTO 1200
     TERM=END
     PRDCSR(TERM)=EXTARC
     PATH_ID(TERM)=.TRUE.
C
C   IF NEGATIVE SURPLUS NODE IS FOUND, DO AN AUGMENTATION
C
     IF (DFCT(TERM).GT.0) GOTO 2000
C
C   RETURN FOR ANOTHER ITERATION
```

```
C
     GO TO 500
    END IF
   ELSE
    EXTARC=-EXTARC
    IF (X(EXTARC).EQ.0) THEN
     SECLEVEL=SB_LEVEL(TERM)
     GO TO 580
    END IF
    START=STARTN(EXTARC)
    BSTLEVEL=P(START)-RC(EXTARC)
    IF (PTERM.GE.BSTLEVEL) THEN
     IF (PATH_ID(START)) GOTO 1200
     TERM=START
     PRDCSR(TERM)=-EXTARC
     PATH_ID(TERM)=.TRUE.
C
C    IF NEGATIVE SURPLUS NODE IS FOUND, DO AN AUGMENTATION
C
     IF (DFCT(TERM).GT.0) GOTO 2000
C
C    RETURN FOR ANOTHER ITERATION
C
     GO TO 500
    END IF
   END IF
C
C    SECOND BEST LOGIC TEST APPLIED TO SAVE A FULL NODE SCAN
C    IF OLD BEST LEVEL CONTINUES TO BE BEST GO FOR ANOTHER
CONTRACTION
C
550  SECLEVEL=SB_LEVEL(TERM)
   IF (BSTLEVEL.LE.SECLEVEL) GOTO 800
C
C    IF SECOND BEST CAN BE USED DO EITHER A CONTRACTION
C    OR START OVER WITH A SPECULATIVE EXTENSION
C
580  IF (SECLEVEL.GT.-LARGE) THEN
    EXTARC=SB_ARC(TERM)
    IF (EXTARC.GT.0) THEN
     IF (U(EXTARC).EQ.0) GOTO 600
     BSTLEVEL=P(ENDN(EXTARC))+RC(EXTARC)
    ELSE
     IF (X(-EXTARC).EQ.0) GOTO 600
     BSTLEVEL=P(STARTN(-EXTARC))-RC(-EXTARC)
    END IF
```

```
      IF (BSTLEVEL.EQ.SECLEVEL) THEN
       SB_LEVEL(TERM)=-LARGE
       EXTEND_ARC(TERM)=EXTARC
       GOTO 800
      END IF
     END IF
C
C   EXTENSION/CONTRACTION ATTEMPT WAS UNSUCCESSFUL, SO SCAN
TERMINAL NODE
C
600  CONTINUE
     NSP=NSP+1

     BSTLEVEL=LARGE
     SECLEVEL=LARGE

     ARC=FPUSHF(TERM)
700  IF (ARC.GT.0) THEN
      NEW_LEVEL = P(ENDN(ARC)) + RC(ARC)
      IF (NEW_LEVEL.LT.SECLEVEL) THEN
       IF (NEW_LEVEL.LT.BSTLEVEL) THEN
        SECLEVEL=BSTLEVEL
        BSTLEVEL=NEW_LEVEL
        SECARC=EXTARC
        EXTARC=ARC
       ELSE
        SECLEVEL=NEW_LEVEL
        SECARC=ARC
       END IF
      END IF
      ARC=NXTPUSHF(ARC)
      GOTO 700
     END IF

     ARC=FPUSHB(TERM)
710  IF (ARC.GT.0) THEN
      NEW_LEVEL = P(STARTN(ARC)) - RC(ARC)
      IF (NEW_LEVEL.LT.SECLEVEL) THEN
       IF (NEW_LEVEL.LT.BSTLEVEL) THEN
        SECLEVEL=BSTLEVEL
        BSTLEVEL=NEW_LEVEL
        SECARC=EXTARC
        EXTARC=-ARC
       ELSE
        SECLEVEL=NEW_LEVEL
        SECARC=-ARC
```

```
          END IF
         END IF
         ARC=NXTPUSHB(ARC)
         GOTO 710
        END IF

        SB_LEVEL(TERM)=SECLEVEL
        SB_ARC(TERM)=SECARC
        EXTEND_ARC(TERM)=EXTARC
C
C    END OF NODE SCAN.
C    IF THE TERMINAL NODE IS THE ROOT, ADJUST ITS PRICE AND CHANGE
ROOT
C
800   IF (TERM.EQ.ROOT) THEN
        P(TERM)=BSTLEVEL+EPS
        IF (PTERM.GE.LARGE) THEN
         PRINT*,'NO PATH TO THE DESTINATION'
         PRINT*,' PROBLEM IS FOUND TO BE INFEASIBLE.'
         PRINT*, 'PROGRAM ENDED; PRESS <CR> TO EXIT'
         PAUSE
         STOP
        END IF
        PATH_ID(ROOT)=.FALSE.
        PREVNODE=ROOT
        ROOT=NXTQUEUE(ROOT)
        GO TO 200
       END IF
C
C    CHECK WHETHER EXTENSION OR CONTRACTION
C
       PRD=PRDCSR(TERM)
       IF (PRD.GT.0) THEN
        PR_TERM=STARTN(PRD)
        PREVLEVEL=P(PR_TERM)-RC(PRD)
       ELSE
        PR_TERM=ENDN(-PRD)
        PREVLEVEL=P(PR_TERM)+RC(-PRD)
       END IF
C
       IF (PREVLEVEL.GT.BSTLEVEL) THEN
C
C    PATH EXTENSION
C
        IF (PREVLEVEL.GE.BSTLEVEL+EPS) THEN
         P(TERM)=BSTLEVEL+EPS
         ELSE
           P(TERM)=PREVLEVEL
         END IF
         IF (EXTARC.GT.0) THEN
           END=ENDN(EXTARC)
           IF (PATH_ID(END)) GOTO 1200
           TERM=END
         ELSE
           START=STARTN(-EXTARC)
           IF (PATH_ID(START)) GOTO 1200
           TERM=START
         END IF
         PRDCSR(TERM)=EXTARC
         PATH_ID(TERM)=.TRUE.
C
C    IF NEGATIVE SURPLUS NODE IS FOUND, DO AN AUGMENTATION
C
       IF (DFCT(TERM).GT.0) GOTO 2000
C
C    RETURN FOR ANOTHER ITERATION
C
        GO TO 500
       ELSE
C
C    PATH CONTRACTION.
C
        P(TERM)=BSTLEVEL+EPS
        PATH_ID(TERM)=.FALSE.
        TERM=PR_TERM
        IF (PR_TERM.NE.ROOT) THEN
          IF (BSTLEVEL.LE.PTERM+EPS) THEN
          GOTO 2000
        END IF
      END IF
      PTERM=P(TERM)
      EXTARC=PRD
      IF (PRD.GT.0) THEN
        BSTLEVEL=BSTLEVEL+EPS+RC(PRD)
      ELSE
        BSTLEVEL=BSTLEVEL+EPS-RC(-PRD)
      END IF
C
C    DO A SECOND BEST TEST AND IF THAT FAILS, DO A FULL NODE SCAN
C
       GOTO 550
      END IF
```

```
C
C    A CYCLE IS ABOUT TO FORM; DO A RETREAT SEQUENCE.
C
1200  CONTINUE
C
      NODE=TERM
1600  IF (NODE.NE.ROOT) THEN
       PATH_ID(NODE)=.FALSE.
       PRD=PRDCSR(NODE)
       IF (PRD.GT.0) THEN
        PR_TERM=STARTN(PRD)
        IF (P(PR_TERM).EQ.P(NODE)+RC(PRD)+EPS) THEN
         NODE=PR_TERM
         GOTO 1600
        END IF
       ELSE
        PR_TERM=ENDN(-PRD)
        IF (P(PR_TERM).EQ.P(NODE)-RC(-PRD)+EPS) THEN
         NODE=PR_TERM
         GOTO 1600
        END IF
       END IF
C
C    DO A FULL SCAN AND PRICE RISE AT PR_TERM
C
       NSP=NSP+1
       BSTLEVEL=LARGE
       SECLEVEL=LARGE
       ARC=FPUSHF(PR_TERM)
1700    IF (ARC.GT.0) THEN
        NEW_LEVEL = P(ENDN(ARC)) + RC(ARC)
        IF (NEW_LEVEL.LT.SECLEVEL) THEN
         IF (NEW_LEVEL.LT.BSTLEVEL) THEN
          SECLEVEL=BSTLEVEL
          BSTLEVEL=NEW_LEVEL
          SECARC=EXTARC
          EXTARC=ARC
         ELSE
          SECLEVEL=NEW_LEVEL
          SECARC=ARC
         END IF
        END IF
        ARC=NXTPUSHF(ARC)
        GOTO 1700
       END IF
C
```

```
       ARC=FPUSHB(PR_TERM)
1710     IF (ARC.GT.0) THEN
          NEW_LEVEL = P(STARTN(ARC)) - RC(ARC)
          IF (NEW_LEVEL.LT.SECLEVEL) THEN
            IF (NEW_LEVEL.LT.BSTLEVEL) THEN
              SECLEVEL=BSTLEVEL
              BSTLEVEL=NEW_LEVEL
              SECARC=EXTARC
              EXTARC=-ARC
            ELSE
              SECLEVEL=NEW_LEVEL
              SECARC=-ARC
            END IF
          END IF
          ARC=NXTPUSHB(ARC)
          GOTO 1710
        END IF

        SB_LEVEL(PR_TERM)=SECLEVEL
        SB_ARC(PR_TERM)=SECARC
        EXTEND_ARC(PR_TERM)=EXTARC

        P(PR_TERM)=BSTLEVEL+EPS
        IF (PR_TERM.EQ.ROOT) THEN
          PREVNODE=ROOT
          PATH_ID(ROOT)=.FALSE.
          ROOT=NXTQUEUE(ROOT)
          GOTO 200
        END IF

        PATH_ID(PR_TERM)=.FALSE.
        PRD=PRDCSR(PR_TERM)
        IF (PRD.GT.0) THEN
          TERM=STARTN(PRD)
        ELSE
          TERM=ENDN(-PRD)
        END IF

        IF (TERM.EQ.ROOT) THEN
          PREVNODE=ROOT
          PATH_ID(ROOT)=.FALSE.
          ROOT=NXTQUEUE(ROOT)
          GOTO 200
        ELSE
          GOTO 2000
        END IF
```

```
      END IF
C
C      END OF AUCTION/SHORTEST PATH ROUTINE.
C      DO AUGMENTATION FROM ROOT AND CORRECT THE PUSH LISTS
C
2000  CONTINUE

      INCR=-DFCT(ROOT)
      NODE = ROOT
2050  EXTARC=EXTEND_ARC(NODE)
      PATH_ID(NODE)=.FALSE.
      IF (EXTARC.GT.0) THEN
       NODE=ENDN(EXTARC)
       IF (INCR.GT.U(EXTARC))  INCR=U(EXTARC)
      ELSE
       NODE=STARTN(-EXTARC)
       IF (INCR.GT.X(-EXTARC))  INCR=X(-EXTARC)
      END IF
      IF (NODE.NE.TERM) GOTO 2050
      PATH_ID(TERM)=.FALSE.
      IF (DFCT(TERM).GT.0) THEN
       IF (INCR.GT.DFCT(TERM)) INCR=DFCT(TERM)
      END IF
C
      NODE = ROOT
2100  EXTARC=EXTEND_ARC(NODE)
      IF (EXTARC.GT.0) THEN
       END=ENDN(EXTARC)
C
C      ADD ARC TO THE REDUCED GRAPH
C
       IF (X(EXTARC).EQ.0) THEN
        NXTPUSHB(EXTARC)=FPUSHB(END)
        FPUSHB(END)=EXTARC
        NEW_LEVEL=P(NODE)-RC(EXTARC)
        IF (SB_LEVEL(END).GT.NEW_LEVEL) THEN
         SB_LEVEL(END)=NEW_LEVEL
         SB_ARC(END)=-EXTARC
        END IF
       END IF
       X(EXTARC)=X(EXTARC)+INCR
       U(EXTARC)=U(EXTARC)-INCR
C
C     REMOVE ARC FROM THE REDUCED GRAPH
C
```

```
      IF (U(EXTARC).EQ.0) THEN
       NAS=NAS+1
       ARC=FPUSHF(NODE)
       IF (ARC.EQ.EXTARC) THEN
        FPUSHF(NODE)=NXTPUSHF(ARC)
       ELSE
        PREVARC=ARC
        ARC=NXTPUSHF(ARC)
2200      IF (ARC.GT.0) THEN
         IF (ARC.EQ.EXTARC) THEN
          NXTPUSHF(PREVARC)=NXTPUSHF(ARC)
          GO TO 2250
         END IF
         PREVARC=ARC
         ARC=NXTPUSHF(ARC)
         GOTO 2200
        END IF
       END IF
      END IF

2250  NODE=END

     ELSE
      EXTARC=-EXTARC
      START=STARTN(EXTARC)
C
C   ADD ARC TO THE REDUCED GRAPH
C
      IF (U(EXTARC).EQ.0) THEN
       NXTPUSHF(EXTARC)=FPUSHF(START)
       FPUSHF(START)=EXTARC
       NEW_LEVEL=P(NODE)+RC(EXTARC)
       IF (SB_LEVEL(START).GT.NEW_LEVEL) THEN
        SB_LEVEL(START)=NEW_LEVEL
        SB_ARC(START)=EXTARC
       END IF
      END IF
      U(EXTARC)=U(EXTARC)+INCR
      X(EXTARC)=X(EXTARC)-INCR
C
C   REMOVE ARC FROM THE REDUCED GRAPH
C
      IF (X(EXTARC).EQ.0) THEN
       NAS=NAS+1
       ARC=FPUSHB(NODE)
       IF (ARC.EQ.EXTARC) THEN
```

```
         FPUSHB(NODE)=NXTPUSHB(ARC)
        ELSE
         PREVARC=ARC
         ARC=NXTPUSHB(ARC)
2300       IF (ARC.GT.0) THEN
          IF (ARC.EQ.EXTARC) THEN
           NXTPUSHB(PREVARC)=NXTPUSHB(ARC)
           GO TO 2350
          END IF
          PREVARC=ARC
          ARC=NXTPUSHB(ARC)
          GOTO 2300
         END IF
        END IF
       END IF

2350     NODE=START

      END IF

      IF (NODE.NE.TERM) GOTO 2100
      DFCT(TERM)=DFCT(TERM)-INCR
      DFCT(ROOT)=DFCT(ROOT)+INCR
C
C     INSERT TERM IN THE QUEUE IF IT HAS A LARGE ENOUGH SURPLUS
C
      IF (DFCT(TERM).LT.THRESH_DFCT) THEN
       IF (NXTQUEUE(TERM).EQ.0) THEN
        NXTNODE=NXTQUEUE(ROOT)
        IF ((P(TERM).GE.P(NXTNODE)).AND.(ROOT.NE.NXTNODE)) THEN
         NXTQUEUE(ROOT)=TERM
         NXTQUEUE(TERM)=NXTNODE
        ELSE
         NXTQUEUE(PREVNODE)=TERM
         NXTQUEUE(TERM)=ROOT
         PREVNODE=TERM
        END IF
       END IF
      END IF
C
C     IF ROOT HAS A LARGE ENOUGH SURPLUS, KEEP IT
C     IN THE QUEUE AND RETURN FOR ANOTHER ITERATION
C
      IF (DFCT(ROOT).LT.THRESH_DFCT) THEN
       PREVNODE=ROOT
       ROOT=NXTQUEUE(ROOT)
```

```
     GO TO 200
    END IF
C
C    END OF AUGMENTATION CYCLE
C
3000  CONTINUE
C
C    CHECK FOR TERMINATION OF SCALING PHASE. IF SCALING PHASE IS
C    NOT FINISHED, ADVANCE THE QUEUE AND RETURN TO TAKE
ANOTHER NODE.
C
    NXTNODE=NXTQUEUE(ROOT)
    IF (ROOT.NE.NXTNODE) THEN
     NXTQUEUE(ROOT)=0
     NXTQUEUE(PREVNODE)=NXTNODE
     ROOT=NXTNODE
     GO TO 200
    END IF
C
C    END OF SUBPROBLEM (SCALING PHASE).
C
3600  CONTINUE
C
C    REDUCE EPSILON.
C
    EPS=INT(EPS/FACTOR)
    IF (EPS.LT.1) EPS=1
    THRESH_DFCT=INT(THRESH_DFCT/FACTOR)
    IF (EPS.EQ.1) THRESH_DFCT=0
C
C    IF ANOTHER AUCTION SCALING PHASE REMAINS, RESET THE FLOWS &
THE PUSH LISTS
C    ELSE RESET ARC FLOWS TO SATISFY CS AND COMPUTE REDUCED
COSTS
C
    IF (CRASH.EQ.1) THEN

     DO 3800 ARC=1,NA
      START=STARTN(ARC)
      END=ENDN(ARC)
      PSTART=P(START)
      PEND=P(END)
      IF (PSTART.GT.PEND+EPS+RC(ARC)) THEN
       RESID=U(ARC)
       IF (RESID.GT.0) THEN
        DFCT(START)=DFCT(START)+RESID
```

```
         DFCT(END)=DFCT(END)-RESID
         X(ARC)=X(ARC)+RESID
         U(ARC)=0
        END IF
       ELSE
        IF (PSTART.LT.PEND-EPS+RC(ARC)) THEN
         FLOW=X(ARC)
         IF (FLOW.GT.0) THEN
          DFCT(START)=DFCT(START)-FLOW
          DFCT(END)=DFCT(END)+FLOW
          X(ARC)=0
          U(ARC)=U(ARC)+FLOW
         END IF
        END IF
       END IF
3800    CONTINUE
C
C     RETURN FOR ANOTHER PHASE
C
3850    CONTINUE
      GOTO 100

     ELSE

      CRASH=1
      DO 3900 ARC=1,NA
       START=STARTN(ARC)
       END=ENDN(ARC)
       RED_COST=RC(ARC)+P(END)-P(START)
       IF (RED_COST.LT.0) THEN
        RESID=U(ARC)
        IF (RESID.GT.0) THEN
         DFCT(START)=DFCT(START)+RESID
         DFCT(END)=DFCT(END)-RESID
         X(ARC)=X(ARC)+RESID
         U(ARC)=0
        END IF
       ELSE
        IF (RED_COST.GT.0) THEN
         FLOW=X(ARC)
         IF (FLOW.GT.0) THEN
          DFCT(START)=DFCT(START)-FLOW
          DFCT(END)=DFCT(END)+FLOW
          X(ARC)=0
          U(ARC)=U(ARC)+FLOW
         END IF
```

```
           END IF
         END IF
         RC(ARC)=RED_COST
3900       CONTINUE

      END IF

      RETURN
      END
C
C
      SUBROUTINE PRINTFLOWS(NODE)
      IMPLICIT INTEGER (A-Z)
C
C---------------------------------------------------------------
C
C  PURPOSE - THIS ROUTINE PRINTS THE DEFICIT AND THE FLOWS
C    OF ARCS INCIDENT TO NODE.  IT IS USED FOR DIAGNOSTIC
C    PURPOSES IN CASE OF AN INFEASIBLE PROBLEM HERE.
C    IT CAN BE USED ALSO FOR MORE GENERAL DIAGNOSTIC
C    PURPOSES.
C
C---------------------------------------------------------------
C
C    MAXNN = DIMENSION OF NODE-LENGTH ARRAYS
C    MAXNA = DIMENSION OF ARC-LENGTH ARRAYS
C
      PARAMETER (MAXNN=10000, MAXNA=70000)
C
      COMMON/ARRAYS/STARTN/ARRAYE/ENDN/ARRAYU/U/ARRAYX/X
     $/ARRAYB/DFCT/BLK3/FOU/BLK4/NXTOU/BLK5/FIN/BLK6/NXTIN
C
      INTEGER
STARTN(MAXNA),ENDN(MAXNA),U(MAXNA),X(MAXNA),DFCT(MAXNN)
      INTEGER FOU(MAXNN),NXTOU(MAXNA)
      INTEGER FIN(MAXNN),NXTIN(MAXNA)
C
C---------------------------------------------------------------
C
      PRINT*,'DEFICIT (I.E., NET FLOW OUT) OF NODE =',DFCT(NODE)
      PRINT*,'FLOWS AND CAPACITIES OF INCIDENT ARCS OF NODE',NODE
C
C    CHECK ALL ARCS LEAVING NODE
C
      IF (FOU(NODE).EQ.0) THEN
        PRINT*,'NO OUTGOING ARCS'
      ELSE
       ARC=FOU(NODE)
 5      IF (ARC.GT.0) THEN
         PRINT*,'ARC',ARC,' BETWEEN NODES',NODE,ENDN(ARC)
         PRINT*,'FLOW =',X(ARC)
         PRINT*,'RESIDUAL CAPACITY =',U(ARC)
         ARC=NXTOU(ARC)
         GO TO 5
        END IF
      END IF
C
C   CHECK ALL ARCS INCOMING TO NODE
C
      IF (FIN(NODE).EQ.0) THEN
       PRINT*,'NO INCOMING ARCS'
      ELSE
       ARC=FIN(NODE)
 10     IF (ARC.GT.0) THEN
         PRINT*,'ARC',ARC,' BETWEEN NODES',STARTN(ARC),NODE
         PRINT*,'FLOW =',X(ARC)
         PRINT*,'RESIDUAL CAPACITY =',U(ARC)
         ARC=NXTIN(ARC)
         GO TO 10
        END IF
      END IF
C
      RETURN
      END
C
C
      SUBROUTINE ASCNT1(DM,DELX,NLABEL,FEASBL,SWITCH,
     $NSCAN,CURNODE,PREVNODE)
      IMPLICIT INTEGER (A-Z)
C
C--------------------------------------------------------------
C
C PURPOSE - THIS SUBROUTINE PERFORMS THE MULTI-NODE PRICE
C   ADJUSTMENT STEP FOR THE CASE WHERE THE SCANNED NODES
C   HAVE POSITIVE DEFICIT.  IT FIRST CHECKS IF DECREASING
C   THE PRICE OF THE SCANNED NODES INCREASES THE DUAL COST.
C   IF YES, THEN IT DECREASES THE PRICE OF ALL SCANNED NODES.
C   THERE ARE TWO POSSIBILITIES FOR PRICE DECREASE:
C   IF SWITCH=.TRUE., THEN THE SET OF SCANNED NODES
C   CORRESPONDS TO AN ELEMENTARY DIRECTION OF MAXIMAL
C   RATE OF ASCENT, IN WHICH CASE THE PRICE OF ALL SCANNED
C   NODES ARE DECREASED UNTIL THE NEXT BREAKPOINT IN THE
```

```
C    DUAL COST IS ENCOUNTERED.  AT THIS POINT, SOME ARC
C    BECOMES BALANCED AND MORE NODE(S) ARE ADDED TO THE
C    LABELED SET AND THE SUBROUTINE IS EXITED.
C    IF SWITCH=.FALSE., THEN THE PRICE OF ALL SCANNED NODES
C    ARE DECREASED UNTIL THE RATE OF ASCENT BECOMES
C    NEGATIVE (THIS CORRESPONDS TO THE PRICE ADJUSTMENT
C    STEP IN WHICH BOTH THE LINE SEARCH AND THE DEGENERATE
C    ASCENT ITERATION ARE IMPLEMENTED).
C
C-----------------------------------------------------------
C
C    MAXNN = DIMENSION OF NODE-LENGTH ARRAYS
C    MAXNA = DIMENSION OF ARC-LENGTH ARRAYS
C
      PARAMETER (MAXNN=10000, MAXNA=70000)
C
C  INPUT PARAMETERS
C
C    DM        = TOTAL DEFICIT OF SCANNED NODES
C    SWITCH    = .TRUE. IF LABELING IS TO CONTINUE AFTER PRICE
CHANGE
C    NSCAN     = NUMBER OF SCANNED NODES
C    CURNODE   = MOST RECENTLY SCANNED NODE
C    N         = NUMBER OF NODES
C    NA        = NUMBER OF ARCS
C    LARGE     = A VERY LARGE INTEGER TO REPRESENT INFINITY
C              (SEE NOTE 3)
C    STARTN(I) = STARTING NODE FOR THE I-TH ARC,    I = 1,...,NA
C    ENDN(I)   = ENDING NODE FOR THE I-TH ARC,      I = 1,...,NA
C    FOU(I)    = FIRST ARC LEAVING I-TH NODE,       I = 1,...,N
C    NXTOU(I)  = NEXT ARC LEAVING THE STARTING NODE OF J-TH ARC,
C                                   I = 1,...,NA
C    FIN(I)    = FIRST ARC ENTERING I-TH NODE,      I = 1,...,N
C    NXTIN(I)  = NEXT ARC ENTERING THE ENDING NODE OF J-TH ARC,
C                                   I = 1,...,NA
C
      INTEGER STARTN(MAXNA),ENDN(MAXNA)
      INTEGER FOU(MAXNN),NXTOU(MAXNA),FIN(MAXNN),NXTIN(MAXNA)
      COMMON /INPUT/N,NA,LARGE
      COMMON /ARRAYS/STARTN/ARRAYE/ENDN
      COMMON /BLK3/FOU/BLK4/NXTOU/BLK5/FIN/BLK6/NXTIN
C
C  UPDATED PARAMETERS
C
C    DELX      = A LOWER ESTIMATE OF THE TOTAL FLOW ON BALANCED
ARCS
```

```
C             IN THE SCANNED-NODES CUT
C    NLABEL    = NUMBER OF LABELED NODES
C    FEASBL    = .FALSE. IF PROBLEM IS FOUND TO BE INFEASIBLE
C    PREVNODE  = THE NODE BEFORE CURNODE IN QUEUE
C    RC(J)     = REDUCED COST OF ARC J,           J = 1,...,NA
C    U(J)      = RESIDUAL CAPACITY OF ARC J,
C                                    J = 1,...,NA
C    X(J)      = FLOW ON ARC J,                   J = 1,...,NA
C    DFCT(I)   = DEFICIT AT NODE I,               I = 1,...,N
C    LABEL(K)  = K-TH NODE LABELED,               K = 1,NLABEL
C    PRDCSR(I) = PREDECESSOR OF NODE I IN TREE OF LABELED NODES
C             (O IF I IS UNLABELED),          I = 1,...,N
C    TFSTOU(I) = FIRST BALANCED ARC OUT OF NODE I,  I = 1,...,N
C    TNXTOU(J) = NEXT BALANCED ARC OUT OF THE STARTING NODE OF
ARC J,
C                                    J = 1,...,NA
C    TFSTIN(I) = FIRST BALANCED ARC INTO NODE I,  I = 1,...,N
C    TNXTIN(J) = NEXT BALANCED ARC INTO THE ENDING NODE OF ARC J,
C                                    J = 1,...,NA
C    NXTQUEUE(I) = NODE FOLLOWING NODE I IN THE FIFO QUEUE
C              (0 IF NODE IS NOT IN THE QUEUE), I = 1,...,N
C    SCAN(I)   = .TRUE. IF NODE I IS SCANNED,      I = 1,...,N
C    MARK(I)   = .TRUE. IF NODE I IS LABELED,      I = 1,...,N
C
      INTEGER RC(MAXNA),U(MAXNA),X(MAXNA),DFCT(MAXNN)
      INTEGER LABEL(MAXNN),PRDCSR(MAXNN)
      INTEGER
TFSTOU(MAXNN),TNXTOU(MAXNA),TFSTIN(MAXNN),TNXTIN(MAXNA)
      INTEGER NXTQUEUE(MAXNN)
      LOGICAL*1 SCAN(MAXNN),MARK(MAXNN)
      COMMON /ARRAYRC/RC/ARRAYU/U/ARRAYX/X/ARRAYB/DFCT
      COMMON /BLK1/LABEL/BLK2/PRDCSR
      COMMON /BLK10/TFSTOU/BLK11/TNXTOU/BLK12/TFSTIN/BLK13/TNXTIN
      COMMON /BLK14/NXTQUEUE
      COMMON /BLK8/SCAN/BLK9/MARK
C
C  WORKING PARAMETERS
C
      INTEGER SAVE(MAXNA)
      LOGICAL*1 SWITCH,FEASBL
      COMMON /BLK7/SAVE
C
C    STORE THE ARCS BETWEEN THE SET OF SCANNED NODES AND
C    ITS COMPLEMENT IN SAVE AND COMPUTE DELPRC, THE STEPSIZE
C    TO THE NEXT BREAKPOINT IN THE DUAL COST IN THE DIRECTION
C    OF DECREASING PRICES OF THE SCANNED NODES.
```

```
C   [THE ARCS ARE STORED INTO SAVE BY LOOKING AT THE ARCS
C   INCIDENT TO EITHER THE SET OF SCANNED NODES OR ITS
C   COMPLEMENT, DEPENDING ON WHETHER NSCAN>N/2 OR NOT.
C   THIS IMPROVES THE EFFICIENCY OF STORING.]
C
      DELPRC=LARGE
      DLX=0
      NSAVE=0
      IF (NSCAN.LE.N/2) THEN
       DO 1 I=1,NSCAN
        NODE=LABEL(I)
        ARC=FOU(NODE)
500       IF (ARC.GT.0) THEN
C
C   ARC POINTS FROM SCANNED NODE TO AN UNSCANNED NODE.
C
         NODE2=ENDN(ARC)
         IF (.NOT.SCAN(NODE2)) THEN
          NSAVE=NSAVE+1
          SAVE(NSAVE)=ARC
          RDCOST=RC(ARC)
          IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE2).NE.ARC))
     $     DLX=DLX+X(ARC)
          IF ((RDCOST.LT.0).AND.(RDCOST.GT.-DELPRC))
     $     DELPRC=-RDCOST
         END IF
         ARC=NXTOU(ARC)
         GOTO 500
        END IF
        ARC=FIN(NODE)

501       IF (ARC.GT.0) THEN
C
C   ARC POINTS FROM UNSCANNED NODE TO SCANNED NODE.
C
         NODE2=STARTN(ARC)
         IF (.NOT.SCAN(NODE2)) THEN
          NSAVE=NSAVE+1
          SAVE(NSAVE)=-ARC
          RDCOST=RC(ARC)
          IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE2).NE.-ARC))
     $     DLX=DLX+U(ARC)
          IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC))
     $     DELPRC=RDCOST
         END IF
         ARC=NXTIN(ARC)
         GOTO 501
        END IF
1      CONTINUE

      ELSE

       DO 2 NODE=1,N
        IF (SCAN(NODE)) GO TO 2
        ARC=FIN(NODE)
502      IF (ARC.GT.0) THEN
         NODE2=STARTN(ARC)
         IF (SCAN(NODE2)) THEN
          NSAVE=NSAVE+1
          SAVE(NSAVE)=ARC
          RDCOST=RC(ARC)
          IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE).NE.ARC))
     $     DLX=DLX+X(ARC)
          IF ((RDCOST.LT.0).AND.(RDCOST.GT.-DELPRC))
     $     DELPRC=-RDCOST
         END IF
         ARC=NXTIN(ARC)
         GOTO 502
        END IF
        ARC=FOU(NODE)
503      IF (ARC.GT.0) THEN
         NODE2=ENDN(ARC)
         IF (SCAN(NODE2)) THEN
          NSAVE=NSAVE+1
          SAVE(NSAVE)=-ARC
          RDCOST=RC(ARC)
          IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE).NE.-ARC))
     $     DLX=DLX+U(ARC)
          IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC))
     $     DELPRC=RDCOST
         END IF
         ARC=NXTOU(ARC)
         GOTO 503
        END IF
2      CONTINUE
      END IF
C
C   CHECK IF THE SET OF SCANNED NODES TRULY CORRESPONDS
C   TO A DUAL ASCENT DIRECTION.  [HERE DELX+DLX IS THE EXACT
C   SUM OF THE FLOW ON ARCS FROM THE SCANNED SET TO THE
C   UNSCANNED SET PLUS THE (CAPACITY - FLOW) ON ARCS FROM
C   THE UNSCANNED SET TO THE SCANNED SET.]
```

```
C      IF THIS WERE NOT THE CASE, SET SWITCH TO .TRUE.
C      AND EXIT SUBROUTINE.
C
      IF ((.NOT.SWITCH).AND.(DELX+DLX.GE.DM)) THEN
       SWITCH=.TRUE.
       RETURN
      END IF
      DELX=DELX+DLX
C
C      CHECK THAT THE PROBLEM IS FEASIBLE.
C
4     IF (DELPRC.EQ.LARGE) THEN
C
C      WE CAN INCREASE THE DUAL COST WITHOUT BOUND, SO
C      THE PRIMAL PROBLEM IS INFEASIBLE.
C
       FEASBL=.FALSE.
       RETURN
      END IF
C
C      DECREASE THE PRICES OF THE SCANNED NODES, ADD MORE
C      NODES TO THE LABELED SET AND CHECK IF A NEWLY LABELED NODE
C      HAS NEGATIVE DEFICIT.
C
      IF (SWITCH) THEN
       DO 7 I=1,NSAVE
        ARC=SAVE(I)
        IF (ARC.GT.0) THEN
         RC(ARC)=RC(ARC)+DELPRC
         IF (RC(ARC).EQ.0) THEN
          NODE2=ENDN(ARC)
          IF (TNXTOU(ARC) .LT. 0) THEN
           TNXTOU(ARC) = TFSTOU(STARTN(ARC))
           TFSTOU(STARTN(ARC)) = ARC
          END IF
          IF (TNXTIN(ARC) .LT. 0) THEN
           TNXTIN(ARC) = TFSTIN(NODE2)
           TFSTIN(NODE2) = ARC
          END IF
          IF (.NOT.MARK(NODE2)) THEN
           PRDCSR(NODE2)=ARC
           NLABEL=NLABEL+1
           LABEL(NLABEL)=NODE2
           MARK(NODE2)=.TRUE.
          END IF
         END IF
```

```
          ELSE
           ARC=-ARC
           RC(ARC)=RC(ARC)-DELPRC
           IF (RC(ARC).EQ.0) THEN
            NODE2=STARTN(ARC)
            IF (TNXTOU(ARC) .LT. 0) THEN
             TNXTOU(ARC) = TFSTOU(NODE2)
             TFSTOU(NODE2) = ARC
            END IF
            IF (TNXTIN(ARC) .LT. 0) THEN
             TNXTIN(ARC) = TFSTIN(ENDN(ARC))
             TFSTIN(ENDN(ARC)) = ARC
            END IF
            IF (.NOT.MARK(NODE2)) THEN
             PRDCSR(NODE2)=-ARC
             NLABEL=NLABEL+1
             LABEL(NLABEL)=NODE2
             MARK(NODE2)=.TRUE.
            END IF
           END IF
          END IF
7        CONTINUE
        RETURN

      ELSE
C
C     DECREASE THE PRICES OF THE SCANNED NODES BY DELPRC.
C     ADJUST FLOW TO MAINTAIN COMPLEMENTARY SLACKNESS WITH
C     THE PRICES.
C
      NB = 0
      DO 6 I=1,NSAVE
       ARC=SAVE(I)
       IF (ARC.GT.0) THEN
        T1=RC(ARC)
        IF (T1.EQ.0) THEN
         T2=X(ARC)
         T3=STARTN(ARC)
         DFCT(T3)=DFCT(T3)-T2
         IF (NXTQUEUE(T3).EQ.0) THEN
          NXTQUEUE(PREVNODE)=T3
          NXTQUEUE(T3)=CURNODE
          PREVNODE=T3
         END IF
         T3=ENDN(ARC)
         DFCT(T3)=DFCT(T3)+T2
```

```
            IF (NXTQUEUE(T3).EQ.0) THEN
             NXTQUEUE(PREVNODE)=T3
             NXTQUEUE(T3)=CURNODE
             PREVNODE=T3
            END IF
            U(ARC)=U(ARC)+T2
            X(ARC)=0
            END IF
            RC(ARC)=T1+DELPRC
            IF (RC(ARC).EQ.0) THEN
             DELX=DELX+X(ARC)
             NB = NB + 1
             PRDCSR(NB) = ARC
            END IF
          ELSE
            ARC=-ARC
            T1=RC(ARC)
            IF (T1.EQ.0) THEN
             T2=U(ARC)
             T3=STARTN(ARC)
             DFCT(T3)=DFCT(T3)+T2
             IF (NXTQUEUE(T3).EQ.0) THEN
              NXTQUEUE(PREVNODE)=T3
              NXTQUEUE(T3)=CURNODE
              PREVNODE=T3
             END IF
             T3=ENDN(ARC)
             DFCT(T3)=DFCT(T3)-T2
             IF (NXTQUEUE(T3).EQ.0) THEN
              NXTQUEUE(PREVNODE)=T3
              NXTQUEUE(T3)=CURNODE
              PREVNODE=T3
             END IF
             X(ARC)=X(ARC)+T2
             U(ARC)=0
           END IF
           RC(ARC)=T1-DELPRC
           IF (RC(ARC).EQ.0) THEN
            DELX=DELX+U(ARC)
            NB = NB + 1
            PRDCSR(NB) = ARC
           END IF
         END IF
6       CONTINUE
      END IF
C
```

```
      IF (DELX.LE.DM) THEN
C
C      THE SET OF SCANNED NODES STILL CORRESPONDS TO A
C      DUAL (POSSIBLY DEGENERATE) ASCENT DIRECTON.  COMPUTE
C      THE STEPSIZE DELPRC TO THE NEXT BREAKPOINT IN THE
C      DUAL COST.
C
       DELPRC=LARGE
       DO 10 I=1,NSAVE
        ARC=SAVE(I)
        IF (ARC.GT.0) THEN
         RDCOST=RC(ARC)
         IF ((RDCOST.LT.0).AND.(RDCOST.GT.-DELPRC)) DELPRC=-RDCOST
        ELSE
         ARC=-ARC
         RDCOST=RC(ARC)
         IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
        END IF
10      CONTINUE
       IF ((DELPRC.NE.LARGE).OR.(DELX.LT.DM)) GO TO 4
      END IF
C
C      ADD NEW BALANCED ARCS TO THE SUPERSET OF BALANCED ARCS.
C
      DO 9 I=1,NB
       ARC=PRDCSR(I)
       IF (TNXTIN(ARC).EQ.-1) THEN
        J=ENDN(ARC)
        TNXTIN(ARC)=TFSTIN(J)
        TFSTIN(J)=ARC
       END IF
       IF (TNXTOU(ARC).EQ.-1) THEN
        J=STARTN(ARC)
        TNXTOU(ARC)=TFSTOU(J)
        TFSTOU(J)=ARC
       END IF
9     CONTINUE

      RETURN
      END
C
C
      SUBROUTINE ASCNT2(DM,DELX,NLABEL,FEASBL,SWITCH,
     $NSCAN,CURNODE,PREVNODE)
      IMPLICIT INTEGER (A-Z)
C
```

```
C---------------------------------------------------------
C
C PURPOSE - THIS ROUTINE IS ANALOGOUS TO ASCNT BUT FOR
C   THE CASE WHERE THE SCANNED NODES HAVE NEGATIVE DEFICIT.
C
C---------------------------------------------------------
C
C   MAXNN = DIMENSION OF NODE-LENGTH ARRAYS
C   MAXNA = DIMENSION OF ARC-LENGTH ARRAYS
C
      PARAMETER (MAXNN=10000, MAXNA=70000)
C
      INTEGER STARTN(MAXNA),ENDN(MAXNA)
      INTEGER FOU(MAXNN),NXTOU(MAXNA),FIN(MAXNN),NXTIN(MAXNA)
      COMMON /INPUT/N,NA,LARGE
      COMMON /ARRAYS/STARTN/ARRAYE/ENDN
      COMMON /BLK3/FOU/BLK4/NXTOU/BLK5/FIN/BLK6/NXTIN
      INTEGER RC(MAXNA),U(MAXNA),X(MAXNA),DFCT(MAXNN)
      INTEGER LABEL(MAXNN),PRDCSR(MAXNN)
      INTEGER
TFSTOU(MAXNN),TNXTOU(MAXNA),TFSTIN(MAXNN),TNXTIN(MAXNA)
      INTEGER NXTQUEUE(MAXNN)
      LOGICAL*1 SCAN(MAXNN),MARK(MAXNN)
      COMMON /ARRAYRC/RC/ARRAYU/U/ARRAYX/X/ARRAYB/DFCT
      COMMON /BLK1/LABEL/BLK2/PRDCSR
      COMMON /BLK10/TFSTOU/BLK11/TNXTOU/BLK12/TFSTIN/BLK13/TNXTIN
      COMMON /BLK14/NXTQUEUE
      COMMON /BLK8/SCAN/BLK9/MARK
      INTEGER SAVE(MAXNA)
      LOGICAL*1 SWITCH,FEASBL
      COMMON /BLK7/SAVE
C
C   STORE THE ARCS BETWEEN THE SET OF SCANNED NODES AND
C   ITS COMPLEMENT IN SAVE AND COMPUTE DELPRC, THE STEPSIZE
C   TO THE NEXT BREAKPOINT IN THE DUAL COST IN THE DIRECTION
C   OF INCREASING PRICES OF THE SCANNED NODES.
C
      DELPRC=LARGE
      DLX=0
      NSAVE=0
      IF (NSCAN.LE.N/2) THEN
       DO 1 I=1,NSCAN
        NODE=LABEL(I)
        ARC=FIN(NODE)
500      IF (ARC.GT.0) THEN
          NODE2=STARTN(ARC)
       IF (.NOT.SCAN(NODE2)) THEN
        NSAVE=NSAVE+1
        SAVE(NSAVE)=ARC
        RDCOST=RC(ARC)
        IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE2).NE.ARC))
   $     DLX=DLX+X(ARC)
        IF ((RDCOST.LT.0).AND.(RDCOST.GT.-DELPRC))
   $     DELPRC=-RDCOST
       END IF
       ARC=NXTIN(ARC)
       GOTO 500
      END IF
      ARC=FOU(NODE)
501    IF (ARC.GT.0) THEN
       NODE2=ENDN(ARC)
       IF (.NOT.SCAN(NODE2)) THEN
        NSAVE=NSAVE+1
        SAVE(NSAVE)=-ARC
        RDCOST=RC(ARC)
        IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE2).NE.-ARC))
   $     DLX=DLX+U(ARC)
        IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC))
   $     DELPRC=RDCOST
       END IF
       ARC=NXTOU(ARC)
       GOTO 501
      END IF
1    CONTINUE
   ELSE
    DO 2 NODE=1,N
      IF (SCAN(NODE)) GO TO 2
      ARC=FOU(NODE)
502    IF (ARC.GT.0) THEN
       NODE2=ENDN(ARC)
       IF (SCAN(NODE2)) THEN
        NSAVE=NSAVE+1
        SAVE(NSAVE)=ARC
        RDCOST=RC(ARC)
        IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE).NE.ARC))
   $     DLX=DLX+X(ARC)
        IF ((RDCOST.LT.0).AND.(RDCOST.GT.-DELPRC))
   $     DELPRC=-RDCOST
       END IF
       ARC=NXTOU(ARC)
       GOTO 502
      END IF
```

```
           ARC=FIN(NODE)
503             IF (ARC.GT.0) THEN
             NODE2=STARTN(ARC)
             IF (SCAN(NODE2)) THEN
               NSAVE=NSAVE+1
               SAVE(NSAVE)=-ARC
               RDCOST=RC(ARC)
               IF ((RDCOST.EQ.0).AND.(PRDCSR(NODE).NE.-ARC))
     $          DLX=DLX+U(ARC)
               IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC))
     $          DELPRC=RDCOST
             END IF
             ARC=NXTIN(ARC)
             GOTO 503
           END IF
2         CONTINUE
      END IF
C
      IF ((.NOT.SWITCH).AND.(DELX+DLX.GE.-DM)) THEN
        SWITCH=.TRUE.
        RETURN
      END IF
      DELX=DELX+DLX
C
C   CHECK THAT THE PROBLEM IS FEASIBLE.
C
4      IF (DELPRC.EQ.LARGE) THEN
        FEASBL=.FALSE.
        RETURN
      END IF
C
C   INCREASE THE PRICES OF THE SCANNED NODES, ADD MORE
C   NODES TO THE LABELED SET AND CHECK IF A NEWLY LABELED NODE
C   HAS POSITIVE DEFICIT.
C
      IF (SWITCH) THEN

       DO 7 I=1,NSAVE
         ARC=SAVE(I)
         IF (ARC.GT.0) THEN
           RC(ARC)=RC(ARC)+DELPRC
           IF (RC(ARC).EQ.0) THEN
             NODE2=STARTN(ARC)
             IF (TNXTOU(ARC) .LT. 0) THEN
               TNXTOU(ARC) = TFSTOU(NODE2)
               TFSTOU(NODE2) = ARC
```

```
          END IF
          IF (TNXTIN(ARC) .LT. 0) THEN
           TNXTIN(ARC) = TFSTIN(ENDN(ARC))
           TFSTIN(ENDN(ARC)) = ARC
          END IF
          IF (.NOT.MARK(NODE2)) THEN
           PRDCSR(NODE2)=ARC
           NLABEL=NLABEL+1
           LABEL(NLABEL)=NODE2
           MARK(NODE2)=.TRUE.
          END IF
         END IF
        ELSE
         ARC=-ARC
         RC(ARC)=RC(ARC)-DELPRC
         IF (RC(ARC).EQ.0) THEN
          NODE2=ENDN(ARC)
          IF (TNXTOU(ARC) .LT. 0) THEN
           TNXTOU(ARC) = TFSTOU(STARTN(ARC))
           TFSTOU(STARTN(ARC)) = ARC
          END IF
          IF (TNXTIN(ARC) .LT. 0) THEN
           TNXTIN(ARC) = TFSTIN(NODE2)
           TFSTIN(NODE2) = ARC
          END IF
          IF (.NOT.MARK(NODE2)) THEN
           PRDCSR(NODE2)=-ARC
           NLABEL=NLABEL+1
           LABEL(NLABEL)=NODE2
           MARK(NODE2)=.TRUE.
          END IF
         END IF
        END IF
  7    CONTINUE
       RETURN

      ELSE

       NB = 0
       DO 6 I=1,NSAVE
        ARC=SAVE(I)
        IF (ARC.GT.0) THEN
         T1=RC(ARC)
         IF (T1.EQ.0) THEN
          T2=X(ARC)
          T3=STARTN(ARC)
  DFCT(T3)=DFCT(T3)-T2
  IF (NXTQUEUE(T3).EQ.0) THEN
   NXTQUEUE(PREVNODE)=T3
   NXTQUEUE(T3)=CURNODE
   PREVNODE=T3
  END IF
  T3=ENDN(ARC)
  DFCT(T3)=DFCT(T3)+T2
  IF (NXTQUEUE(T3).EQ.0) THEN
   NXTQUEUE(PREVNODE)=T3
   NXTQUEUE(T3)=CURNODE
   PREVNODE=T3
  END IF
  U(ARC)=U(ARC)+T2
  X(ARC)=0
 END IF
 RC(ARC)=T1+DELPRC
 IF (RC(ARC).EQ.0) THEN
  DELX=DELX+X(ARC)
  NB = NB + 1
  PRDCSR(NB) = ARC
 END IF
ELSE
 ARC=-ARC
 T1=RC(ARC)
 IF (T1.EQ.0) THEN
  T2=U(ARC)
  T3=STARTN(ARC)
  DFCT(T3)=DFCT(T3)+T2
  IF (NXTQUEUE(T3).EQ.0) THEN
   NXTQUEUE(PREVNODE)=T3
   NXTQUEUE(T3)=CURNODE
   PREVNODE=T3
  END IF
  T3=ENDN(ARC)
  DFCT(T3)=DFCT(T3)-T2
  IF (NXTQUEUE(T3).EQ.0) THEN
   NXTQUEUE(PREVNODE)=T3
   NXTQUEUE(T3)=CURNODE
   PREVNODE=T3
  END IF
  X(ARC)=X(ARC)+T2
  U(ARC)=0
 END IF
 RC(ARC)=T1-DELPRC
 IF (RC(ARC).EQ.0) THEN
```

```
          DELX=DELX+U(ARC)
          NB = NB + 1
          PRDCSR(NB) = ARC
         END IF
        END IF
6       CONTINUE

      END IF
C
      IF (DELX.LE.-DM) THEN
       DELPRC=LARGE
       DO 10 I=1,NSAVE
        ARC=SAVE(I)
        IF (ARC.GT.0) THEN
         RDCOST=RC(ARC)
         IF ((RDCOST.LT.0).AND.(RDCOST.GT.-DELPRC)) DELPRC=-RDCOST
        ELSE
         ARC=-ARC
         RDCOST=RC(ARC)
         IF ((RDCOST.GT.0).AND.(RDCOST.LT.DELPRC)) DELPRC=RDCOST
        END IF
10      CONTINUE
       IF ((DELPRC.NE.LARGE).OR.(DELX.LT.-DM)) GO TO 4
      END IF
C
C      ADD NEW BALANCED ARCS TO THE SUPERSET OF BALANCED ARCS.
C
      DO 9 I=1,NB
       ARC=PRDCSR(I)
       IF (TNXTIN(ARC).EQ.-1) THEN
        J=ENDN(ARC)
        TNXTIN(ARC)=TFSTIN(J)
        TFSTIN(J)=ARC
       END IF
       IF (TNXTOU(ARC).EQ.-1) THEN
        J=STARTN(ARC)
        TNXTOU(ARC)=TFSTOU(J)
        TFSTOU(J)=ARC
       END IF
9     CONTINUE

      RETURN
      END
C
C
      SUBROUTINE SENSTV
      IMPLICIT INTEGER (A-Z)
C
C------------------------------------------------------------------
C
C PURPOSE - THIS SUBROUTINE ALLOWS THE USER TO INTERACTIVELY
C   EITHER CHANGE NODE SUPPLY, OR CHANGE FLOW UPPER BOUND
C   OF AN EXISTING ARC, OR CHANGE COST OF AN EXISTING ARC,
C   OR DELETE AN EXISTING ARC, OR ADD AN ARC.
C   [NOTE: IF THE OPERATING SYSTEM RESETS ALL LOCAL VARIABLES
C   TO SOME DEFAULT VALUE EACH TIME THIS SUBROUTINE IS CALLED,
C   THEN THE USER MUST MAKE THE FOLLOWING CURRENTLY LOCAL
C   VARIABLES:
C       DELARC, DARC, DU, ADDARC, AARC
C   GLOBAL (BY EITHER PUTTING THEM IN A COMMON BLOCK OR
C   PASSING THEM THROUGH THE CALLING PARAMETER).]
C
C------------------------------------------------------------------
C
C   MAXNN = DIMENSION OF NODE-LENGTH ARRAYS
C   MAXNA = DIMENSION OF ARC-LENGTH ARRAYS
C
      PARAMETER (MAXNN=10000, MAXNA=70000)
C
C INPUT PARAMETERS
C
C   N         = NUMBER OF NODES
C   NA        = NUMBER OF ARCS
C   LARGE     = A VERY LARGE INTEGER TO REPRESENT INFINITY
C   STARTN(J) = STARTING NODE FOR ARC J,          J = 1,...,NA
C   ENDN(J)   = ENDING NODE FOR ARC J,            J = 1,...,NA
C   FOU(I)    = FIRST ARC OUT OF NODE I,          I = 1,...,N
C   NXTOU(J)  = NEXT ARC OUT OF THE STARTING NODE OF ARC J,
C                                   J = 1,...,NA
C   FIN(I)    = FIRST ARC INTO NODE I,            I = 1,...,N
C   NXTIN(J)  = NEXT ARC INTO THE ENDING NODE OF ARC J,
C                                   J = 1,...,NA
C   REPEAT    = .TRUE. IF CAP(J)=X(J)+U(J), J=1,...,NA, ON INPUT
C             (.FALSE. OTHERWISE)
C
      INTEGER STARTN(MAXNA),ENDN(MAXNA)
      INTEGER FOU(MAXNN),NXTOU(MAXNA),FIN(MAXNN),NXTIN(MAXNA)
      LOGICAL*1 REPEAT
      COMMON /INPUT/N,NA,LARGE
      COMMON /ARRAYS/STARTN/ARRAYE/ENDN
      COMMON /BLK3/FOU/BLK4/NXTOU/BLK5/FIN/BLK6/NXTIN
      COMMON /BLKR/REPEAT
```

```
C
C  UPDATED PARAMETERS
C
C    C(J)      = COST OF ARC J,                    J = 1,...,NA
C    CAP(J)    = CAPACITY OF ARC J,                J = 1,...,NA
C    RC(J)     = REDUCED COST OF ARC J,            J = 1,...,NA
C    X(J)      = FLOW ON ARC J,                    J = 1,...,NA
C    U(J)      = CAP(J) - X(J) ON OUTPUT,          J = 1,...,NA
C    DFCT(I)   = DEMAND AT NODE I ON INPUT
C                AND ZERO ON OUTPUT,               I = 1,...,N
C    TFSTOU(I) = FIRST BALANCED ARC OUT OF NODE I,  I = 1,...,N
C    TNXTOU(J) = NEXT BALANCED ARC OUT OF THE STARTING NODE OF
ARC J,
C                                  J = 1,...,NA
C    TFSTIN(I) = FIRST BALANCED ARC INTO NODE I,  I = 1,...,N
C    TNXTIN(J) = NEXT BALANCED ARC INTO THE ENDING NODE OF ARC J,
C                                  J = 1,...,NA
C
     INTEGER C(MAXNA),CAP(MAXNA)
     INTEGER RC(MAXNA),X(MAXNA),U(MAXNA),DFCT(MAXNN)
     INTEGER
TFSTOU(MAXNN),TNXTOU(MAXNA),TFSTIN(MAXNN),TNXTIN(MAXNA)
     COMMON /ARRAYC/C/BLKCAP/CAP
     COMMON /ARRAYRC/RC/ARRAYX/X/ARRAYU/U/ARRAYB/DFCT
     COMMON/BLK10/TFSTOU/BLK11/TNXTOU/BLK12/TFSTIN/BLK13/TNXTIN
C
C  WORKING PARAMETERS
C
     INTEGER LABEL(MAXNN),PRICE(MAXNN),SAVE(MAXNA)
     LOGICAL*1 ADDARC,DELARC,MARK(MAXNN)
     COMMON/BLK1/LABEL/BLK2/PRICE/BLK7/SAVE
     COMMON/BLK9/MARK
C
     IF (.NOT.REPEAT) THEN
C
C    RESTORE THE ARC CAPACITY TO THAT OF THE ORIGINAL PROBLEM
C    [RECALL THAT, IN SOLVING THE ORIGINAL PROBLEM, RELAX4
C    MAY HAVE DECREASED THE ARC CAPACITIES DURING
C    INITIALIZATION PHASE I.]  THEN UPDATE FLOW AND THE
C    NODE DEFICITS TO MATCH THIS "NEW" CAPACITY.
C
      DO 10 I=1,NA
        IF (RC(I).LT.0) THEN
          DFCT(STARTN(I))=DFCT(STARTN(I))+CAP(I)-X(I)
          DFCT(ENDN(I))=DFCT(ENDN(I))-CAP(I)+X(I)
          X(I)=CAP(I)
```

```
       ELSE
        U(I)=CAP(I)-X(I)
       END IF
10     CONTINUE
      REPEAT=.TRUE.
     END IF
20     WRITE(6,30)
      WRITE(6,40)
      WRITE(6,50)
      WRITE(6,60)
      WRITE(6,70)
      WRITE(6,80)
      IF (ADDARC) WRITE(6,90) AARC
      IF (DELARC) WRITE(6,100) DARC
      WRITE(6,105)
30     FORMAT(' INPUT 0 TO SOLVE THE MODIFIED PROBLEM')
40     FORMAT('      1 TO CHANGE NODE FLOW SUPPLY')
50     FORMAT('      2 TO CHANGE ARC FLOW UPPER BOUND')
60     FORMAT('      3 TO CHANGE ARC COST')
70     FORMAT('      4 TO DELETE AN ARC')
80     FORMAT('      5 TO ADD AN ARC')
90     FORMAT('      6 TO DELETE LAST ARC',I8,' ADDED')
100     FORMAT('      7 TO RESTORE LAST ARC',I8,' DELETED')
105     FORMAT('      8 TO QUIT PROGRAM')
      READ(5,*)SEL
      IF (SEL.EQ.8) THEN
       STOP
      ELSE IF (SEL.EQ.0) THEN
       RETURN
      ELSE IF (SEL.EQ.1) THEN
C
C    CHANGE NODE FLOW SUPPLY.
C
110      WRITE(6,120)
120      FORMAT(' INPUT NODE # WHERE FLOW SUPPLY IS INCREASED')
       READ(5,*)NODE
       IF ((NODE.LE.0).OR.(NODE.GT.N)) GO TO 110
       WRITE(6,130)
130      FORMAT(' INPUT AMOUNT OF INCREASE (<0 VALUE ALLOWED)')
       READ(5,*)DELSUP
       DFCT(NODE)=DFCT(NODE)-DELSUP
140      WRITE(6,150)
150      FORMAT(' INPUT NODE NO. WHERE FLOW SUPPLY IS DECREASED')
       READ(5,*)NODE
       IF ((NODE.LE.0).OR.(NODE.GT.N)) GO TO 140
       DFCT(NODE)=DFCT(NODE)+DELSUP
      ELSE IF (SEL.EQ.2) THEN
C
C     CHANGE ARC FLOW CAPACITY.
C     [NOTE: U IS THE RESIDUAL CAPACITY, I.E., U = CAPACITY - X.]
C
160      WRITE(6,170)
170      FORMAT(' INPUT ARC NO. AND THE INCREASE IN UPPER BOUND')
       READ(5,*)ARC,DELUB
       IF ((ARC.LE.0).OR.(ARC.GT.NA)) GO TO 160
       IF (RC(ARC).LT.0) THEN
C
C     ARC IS ACTIVE, SO MAINTAIN FLOW AT (NEW) CAPACITY.
C
        DFCT(STARTN(ARC))=DFCT(STARTN(ARC))+DELUB
        DFCT(ENDN(ARC))=DFCT(ENDN(ARC))-DELUB
        X(ARC)=X(ARC)+DELUB
        IF (X(ARC).LT.0) WRITE(6,180)
       ELSE IF (RC(ARC).EQ.0) THEN
        IF (U(ARC).GE.-DELUB) THEN
         U(ARC)=U(ARC)+DELUB
        ELSE
C
C     NEW CAPACITY IS LESS THAN CURRENT FLOW, SO DECREASE
C     FLOW TO NEW CAPACITY.
C
         DEL=-DELUB-U(ARC)
         DFCT(STARTN(ARC))=DFCT(STARTN(ARC))-DEL
         DFCT(ENDN(ARC))=DFCT(ENDN(ARC))+DEL
         X(ARC)=X(ARC)-DEL
         IF (X(ARC).LT.0) WRITE(6,180)
         U(ARC)=0
        END IF
       ELSE
        U(ARC)=U(ARC)+DELUB
        IF (U(ARC).LT.0) WRITE(6,180)
180       FORMAT(' FLOW UPPER BOUND IS NOW < 0')
       END IF
      ELSE IF (SEL.EQ.3) THEN
C
C     CHANGE ARC COST.
C
190      WRITE(6,200)
200      FORMAT(' INPUT ARC NO. & INCREASE IN COST')
       READ(5,*)ARC,DELC
       IF ((ARC.LE.0).OR.(ARC.GT.NA)) GO TO 190
       IF ((RC(ARC).GE.0).AND.(RC(ARC)+DELC.LT.0)) THEN
```

```
C
C    ARC BECOMES ACTIVE, SO INCREASE FLOW TO CAPACITY.
C
          DFCT(STARTN(ARC))=DFCT(STARTN(ARC))+U(ARC)
          DFCT(ENDN(ARC))=DFCT(ENDN(ARC))-U(ARC)
          X(ARC)=U(ARC)+X(ARC)
          U(ARC)=0
        ELSE IF ((RC(ARC).LE.0).AND.(RC(ARC)+DELC.GT.0))THEN
C
C    ARC BECOMES INACTIVE, SO DECREASE FLOW TO ZERO.
C
          DFCT(STARTN(ARC))=DFCT(STARTN(ARC))-X(ARC)
          DFCT(ENDN(ARC))=DFCT(ENDN(ARC))+X(ARC)
          U(ARC)=U(ARC)+X(ARC)
          X(ARC)=0
        END IF
        RC(ARC)=RC(ARC)+DELC
        C(ARC)=C(ARC)+DELC
C
C   IF ARC BECOMES BALANCED, CHECK TO ADD ARC TO TFSTOU,
TFSTIN,....
C
        IF ((RC(ARC).EQ.0).AND.(DELC.NE.0)) THEN
          CALL ADDTR(ARC)
        END IF
C
      ELSE IF ((SEL.EQ.4).OR.(SEL.EQ.6)) THEN
C
C    DELETE AN ARC.
C
        IF (SEL.EQ.6) THEN
          IF (.NOT.ADDARC) GO TO 20
          ADDARC=.FALSE.
          ARC=AARC
        ELSE
210         WRITE(6,220)
220         FORMAT(' INPUT ARC NO. FOR DELETION')
          READ(5,*)ARC
          IF ((ARC.LE.0).OR.(ARC.GT.NA)) GO TO 210
          DELARC=.TRUE.
          DARC=ARC
          DU=U(ARC)+X(ARC)
        END IF
C
C    REMOVE ARC FROM THE ARRAY FIN, FOU, NXTIN, NXTOU.
C
```

```
      ARC1=FOU(STARTN(ARC))
      IF (ARC1.EQ.ARC) THEN
       FOU(STARTN(ARC))=NXTOU(ARC1)
      ELSE
230       ARC2=NXTOU(ARC1)
       IF (ARC2.EQ.ARC) THEN
        NXTOU(ARC1)=NXTOU(ARC2)
        GO TO 240
       END IF
       ARC1=ARC2
       IF (NXTOU(ARC1).GT.0) GO TO 230
      END IF
240    ARC1=FIN(ENDN(ARC))
      IF (ARC1.EQ.ARC) THEN
       FIN(ENDN(ARC))=NXTIN(ARC1)
      ELSE
250       ARC2=NXTIN(ARC1)
       IF (ARC2.EQ.ARC) THEN
        NXTIN(ARC1)=NXTIN(ARC2)
        GO TO 260
       END IF
       ARC1=ARC2
       IF (NXTIN(ARC1).GT.0) GO TO 250
      END IF
C
C    REMOVE ARC FROM THE ARRAY TFSTIN, TFSTOU, TNXTIN, TNXTOU.
C
260    ARC1=TFSTOU(STARTN(ARC))
      IF (ARC1.EQ.0) GO TO 262
      IF (ARC1.EQ.ARC) THEN
       TFSTOU(STARTN(ARC))=TNXTOU(ARC1)
      ELSE
261       ARC2=TNXTOU(ARC1)
       IF (ARC2.EQ.ARC) THEN
        TNXTOU(ARC1)=TNXTOU(ARC2)
        GO TO 262
       END IF
       ARC1=ARC2
       IF (TNXTOU(ARC1).GT.0) GO TO 261
      END IF
262    ARC1=TFSTIN(ENDN(ARC))
      IF (ARC1.EQ.0) GO TO 264
      IF (ARC1.EQ.ARC) THEN
       TFSTIN(ENDN(ARC))=TNXTIN(ARC1)
      ELSE
263       ARC2=TNXTIN(ARC1)
```

```
          IF (ARC2.EQ.ARC) THEN
           TNXTIN(ARC1)=TNXTIN(ARC2)
           GO TO 264
          END IF
          ARC1=ARC2
          IF (TNXTIN(ARC1).GT.0) GO TO 263
         END IF
264       TNXTOU(ARC) = -1
         TNXTIN(ARC) = -1
C
C    REMOVE FLOW OF ARC FROM NETWORK BY SETTING ITS FLOW
C    AND CAPACITY TO 0.
C
         DFCT(STARTN(ARC))=DFCT(STARTN(ARC))-X(ARC)
         DFCT(ENDN(ARC))=DFCT(ENDN(ARC))+X(ARC)
         X(ARC)=0
         U(ARC)=0
        ELSE IF ((SEL.EQ.5).OR.(SEL.EQ.7)) THEN
         IF (SEL.EQ.7) THEN
          IF (.NOT.DELARC) GO TO 20
          IARC=DARC
          IH=STARTN(IARC)
          IT=ENDN(IARC)
          DELARC=.FALSE.
          IU=DU
         ELSE
270        WRITE(6,280)NA+1
280       FORMAT(' INPUT HEAD & TAIL NODES OF NEW ARC',I8)
         READ(5,*)IH,IT
         IF ((IH.LE.0).OR.(IH.GT.N).OR.(IT.LE.0).OR.(IT.GT.N))GO TO 270
290        WRITE(6,300)
300       FORMAT(' INPUT COST & FLOW UPPER BD')
         READ(5,*)IC,IU
         IF (IU.LT.0) GO TO 290
         ADDARC=.TRUE.
         AARC=NA+1
         NA=NA+1
         C(NA)=IC
         STARTN(NA)=IH
         ENDN(NA)=IT
         IARC=NA
        END IF
C
C    DETERMINE THE DUAL PRICES AT IH AND IT:
C    FIRST SET THE PRICE AT IH TO ZERO AND THEN CONSTRUCT THE
C    PRICE AT OTHER NODES BY BREADTH-FIRST SEARCH AND USING
```

```
C    THE FACT THAT
C    RC(ARC) = C(ARC) - PRICE(STARTN(ARC)) + PRICE(ENDN(ARC)).
C    THE NETWORK (GIVEN BY FOU, NXTOU, FIN, NXTIN) NEED NOT BE
CONNECTED.
C
      NSCAN=0
      NLABEL=1
      LABEL(1)=IH
      PRICE(IH)=0
      DO 310 I=1,N
310     MARK(I)=.FALSE.
      MARK(IH)=.TRUE.
320     IF (NLABEL.GT.NSCAN) THEN
        NSCAN=NSCAN+1
        NODE=LABEL(NSCAN)
        ARC=FOU(NODE)
330       IF (ARC.GT.0) THEN
          NODE2=ENDN(ARC)
          IF (.NOT.MARK(NODE2)) THEN
            MARK(NODE2)=.TRUE.
            PRICE(NODE2)=RC(ARC)-C(ARC)+PRICE(NODE)
            IF (NODE2.EQ.IT) GO TO 370
            NLABEL=NLABEL+1
            LABEL(NLABEL)=NODE2
          END IF
          ARC=NXTOU(ARC)
          GO TO 330
        END IF
        ARC=FIN(NODE)
340       IF (ARC.GT.0) THEN
          NODE2=STARTN(ARC)
          IF (.NOT.MARK(NODE2)) THEN
            MARK(NODE2)=.TRUE.
            PRICE(NODE2)=C(ARC)-RC(ARC)+PRICE(NODE)
            IF (NODE2.EQ.IT) GO TO 370
            NLABEL=NLABEL+1
            LABEL(NLABEL)=NODE2
          END IF
          ARC=NXTIN(ARC)
          GO TO 340
        END IF
        GO TO 320
      END IF
      PRICE(IT)=-C(IARC)
C
```

```
C    COMPUTE REDUCED COST OF THE NEW ARC AND UPDATE FLOW AND
DEFICIT
C    ACCORDINGLY.
C
370    RC(IARC)=C(IARC)+PRICE(IT)
     IF (RC(IARC).LT.0) THEN
      DFCT(IH)=DFCT(IH)+IU
      DFCT(IT)=DFCT(IT)-IU
      X(IARC)=IU
      U(IARC)=0
     ELSE
      X(IARC)=0
      U(IARC)=IU
     END IF
     NXTOU(IARC)=FOU(IH)
     FOU(IH)=IARC
     NXTIN(IARC)=FIN(IT)
     FIN(IT)=IARC
     IF (RC(IARC).EQ.0) THEN
      TNXTOU(IARC)=TFSTOU(IH)
      TFSTOU(IH)=IARC
      TNXTIN(IARC)=TFSTIN(IT)
      TFSTIN(IT)=IARC
     END IF
    END IF
    GO TO 20
    END
C
C
    SUBROUTINE ADDTR(ARC)
    IMPLICIT INTEGER (A-Z)
C
C-----------------------------------------------------------
C
C  PURPOSE - THIS SUBROUTINE CHECKS IF ARC IS IN THE ARRAYS
TFSTOU, TNXTOU,
C    TFSTIN, TNXTIN AND, IF NOT, ADDS ARC TO THEM.
C
C-----------------------------------------------------------
C
C    MAXNN = DIMENSION OF NODE-LENGTH ARRAYS
C    MAXNA = DIMENSION OF ARC-LENGTH ARRAYS
C
    PARAMETER (MAXNN=10000, MAXNA=70000)
C
    COMMON/ARRAYS/STARTN/ARRAYE/ENDN
```

```
   COMMON/BLK10/TFSTOU/BLK11/TNXTOU/BLK12/TFSTIN/BLK13/TNXTIN
   INTEGER STARTN(MAXNA),ENDN(MAXNA)
   INTEGER
TFSTOU(MAXNN),TNXTOU(MAXNA),TFSTIN(MAXNN),TNXTIN(MAXNA)
C
   NODE=STARTN(ARC)
   ARC1=TFSTOU(NODE)
10  IF (ARC1.GT.0) THEN
    IF (ARC1.EQ.ARC) GO TO 20
    ARC1=TNXTOU(ARC1)
    GO TO 10
   END IF
   TNXTOU(ARC)=TFSTOU(NODE)
   TFSTOU(NODE)=ARC
20  NODE=ENDN(ARC)
   ARC1=TFSTIN(NODE)
30  IF (ARC1.GT.0) THEN
    IF (ARC1.EQ.ARC) RETURN
    ARC1=TNXTIN(ARC1)
    GO TO 30
   END IF
   TNXTIN(ARC)=TFSTIN(NODE)
   TFSTIN(NODE)=ARC
   RETURN
   END
C
```

GRAPHICAL USER INTERFACE CODE

The following codes are individual files that use the output from the Relax-IV solution set to produce the output for the OffSite software.

```
/*---------------------------------------------------------
  statbar.cpp
 ---------------------------------------------------------*/

#include <windows.h>
#include <commctrl.h>

DWORD dwStatusBarStyles = WS_CHILD |
                    WS_VISIBLE |
                    WS_CLIPSIBLINGS |
                    CCS_BOTTOM |
                    SBARS_SIZEGRIP ;

HWND init_status_bar (HWND hwndParent)
{
  HWND hwndSB ;

  hwndSB = CreateStatusWindow (dwStatusBarStyles,
                          "",
                          hwndParent,
                          2) ;
  return hwndSB ;
}

#ifndef __conf_macros_h__
#define __conf_macros_h__

#define min(a,b)  ((a)<(b) ? (a) : (b))
#define max(a,b)  ((a)>(b) ? (a) : (b))

#endif

void lat_long_to_x_y(
                    double origin_lat,  // in decimal degrees
                    double origin_long,  // in decimal degrees
                    double pt_lat,       // lat of a point in decimal deg
                    double pt_long,      // long of a point in decimal deg
                    double *x_coord,
                    double *y_coord
      );

void rhumbline_latlong_to_range_bear(
            double lat1,  // 1st pt latitude in decimal degrees
                   double long1,  // 1st pt longitude in decimal degrees
            double lat2,  // 2nd pt latitude in decimal degrees
                   double long2,  // 2nd pt longitude in decimal degrees
                   double *range,   // in kilometers
            double *bearing  // from pt 1 to pt 2 in degrees
            );

#include <math.h>
#include "constants.h"
#include "lat_long.h"

/*
 Function:    lat_long_to_x_y

 File:        lat_long.cpp

 Description: lat_long_to_x_y converts a specific latitude and longitude
              to rectangular coordinates x and y.
```

```
*/

void lat_long_to_x_y(
                    double origin_lat,  // in decimal degrees
                    double origin_long, // in decimal degrees
                    double pt_lat,      // lat of a point in decimal deg
                    double pt_long,     // long of a point in decimal deg
                    double *x_coord,
                    double *y_coord)
{
  double distance;
  double bearing;
  double theta;

  // Get the distance and bearing from the origin to the point

  rhumbline_latlong_to_range_bear(origin_lat, origin_long, pt_lat, pt_long,
                                  &distance, &bearing);

  // Convert direction from bearing to standard notation measured
  // counterclockwise from east

  theta = 450. - bearing;

  if ( theta >= 360. ) theta -= 360.;

  theta /= DEG_PER_RAD;  // convert direction to radians

  // Convert polar coordinates to rectangular coordinates

  *x_coord = distance * cos(theta);
  *y_coord = distance * sin(theta);

} // end of lat_long_to_x_y

/*
  Function:    rhumbline_latlong_to_range_bear

  Description: rhumbline_latlong_to_range_bear calculates the rhumbline
     distance and bearing between two points.
*/

void rhumbline_latlong_to_range_bear(
            double lat1,  // 1st pt latitude in decimal degrees
                          double long1,  // 1st pt longitude in decimal degrees
            double lat2,  // 2nd pt latitude in decimal degrees
```

```
                        double long2,  // 2nd pt longitude in decimal degrees
                        double *range,   // in kilometers
                double *bearing) // from pt 1 to pt 2 in degrees
{
 double delta_lat;
 double delta_long;
 double cosine_lat1;
 double cosine_lat2;
 double delta_y;
 double delta_x;
 double prelim_delta_x;

 delta_lat = lat2 - lat1;
 delta_long = long2 - long1;

 // Convert the input coordinates to radians

 lat1 /= DEG_PER_RAD;
 lat2 /= DEG_PER_RAD;
 long1 /= DEG_PER_RAD;
 long2 /= DEG_PER_RAD;

 if (delta_lat == 0. && delta_long == 0.)
   {
   *range = 0.;
   *bearing = 0.;
   return;
   }

 cosine_lat1 = cos(lat1);
 cosine_lat2 = cos(lat2);

 if (delta_long > 180.0)
   {
   delta_long -= TWO_PI;
   }
 else
   {
   if (delta_long < -180.0)
            delta_long += TWO_PI;
  }

 // Convert from degrees to nautical miles

 delta_y = delta_lat * NM_PER_DEG;
 prelim_delta_x = delta_long * NM_PER_DEG;
```

```
    // Correct for longitude

    delta_x = prelim_delta_x * ((cosine_lat1 + cosine_lat2) * 0.5);

    // Calculate the range and convert to Kilometers

    *range = sqrt((delta_x * delta_x) + (delta_y * delta_y));
    *range *= KM_PER_NM;

    // Calculate the bearing

    if (delta_y != 0.)
      {
      *bearing = atan2(delta_x, delta_y);
      }
    else
      {
      if (delta_x > 0.)
              *bearing = M_PI / 2.0;
      else
              *bearing = -M_PI / 2.0;
      }

    // Convert to degrees

    *bearing *= DEG_PER_RAD;

    if (*bearing < 0.)
      *bearing += 360.0;

} // end of rhumbline_latlong_to_range_bear

/*
  FILE:       TLIST.H

  DESCRIPTION:  This file contains the generic class definition of the doubly
            linked list.
*/

#ifndef __tlist_h__
#define __tlist_h__

template <class T> class _CLASSTYPE TEntry
{
public:
  T       *Data;
  TEntry  *Next;
  TEntry  *Previous;

  TEntry(T *);
  virtual ~TEntry(void);
};

template <class T> class _CLASSTYPE TList
{
protected:
  int       NumberEntries;
  TEntry<T>  *head;
  TEntry<T>  *tail;

public:
  TList(void);
  virtual ~TList(void);
  virtual void AddEntry(T *);
  virtual void ClearList(void);
  void Initialize(void);
  int GetNumberEntries(void) { return NumberEntries; } ;
  virtual T *GetEntry(int);
  virtual T *RemoveEntry(TEntry<T> *);
  virtual T *RemoveNextEntry(void);
};

template <class T> class _CLASSTYPE TPathList : public TList<T>
{
public:
  virtual void Draw(int, int, HDC, BOOL, HPEN *);
};

template <class T> class _CLASSTYPE TMapEntityList : public TList<T>
{
public:
  virtual void Draw(HDC, BOOL, HPEN *);
};

//-----------------------------------------------
// TEntry's method implementations:
//-----------------------------------------------

template <class T> TEntry<T>::TEntry(T *new_data)
{
  Data = new_data;
```

```
    Next = NULL;
    Previous = NULL;
  }

  template <class T> TEntry<T>::~TEntry(void)
  {
  }

  //-----------------------------------------------
  // TList's method implementations:
  //-----------------------------------------------

  template <class T> TList<T>::TList(void)
  {
    // Initialize the list

    head = NULL;
    tail = NULL;
    NumberEntries = 0;
  }

  template <class T> TList<T>::~TList(void)
  {
    ClearList();
  }

  template <class T> void TList<T>::ClearList(void)
  {
    T *the_data;

    // Clear the list

    while ((the_data = RemoveNextEntry()) != NULL)
    {
      delete the_data;
    }
  }

  template <class T> void TList<T>::Initialize(void)
  {
    // Initialize the list

    head = NULL;
    tail = NULL;
    NumberEntries = 0;
  }
```

```
template <class T> T  *TList<T>::GetEntry(int the_position)
{
 int      i;         // current position in the list
 TEntry<T> *the_entry; // pointer for traversing the list

 i = 1;
 the_entry = head;

 while(i < the_position)
   {
   i++;
   the_entry = the_entry->Next;
   }

 return(the_entry->Data);
}

template <class T> T *TList<T>::RemoveEntry(TEntry<T> *del_entry)
{
 T *the_data;

 if (del_entry == NULL)
   {
   the_data = NULL;
   }
 else
   {
   the_data = del_entry->Data;

   if ( del_entry->Previous != NULL )
     del_entry->Previous->Next = del_entry->Next;
   else
     head = del_entry->Next;

   if ( del_entry->Next != NULL )
           del_entry->Next->Previous = del_entry->Previous;
   else
           tail = del_entry->Previous;

   // The list's links are in place, free the removed entry and decrement
   // the counter.

   delete del_entry;
   --NumberEntries;
   }
```

```
  return(the_data);

}

template <class T> T *TList<T>::RemoveNextEntry(void)
{
  TEntry<T> *the_entry; // next entry to return
  T         *the_data;

  // Get the first entry in the list

  the_entry = head;

  // If the list is empty, return a NULL pointer

  if (the_entry == NULL)
    {
    the_data = NULL;
    }
  else
    {
    the_data = the_entry->Data;

    // If there are more entries in the list, the next entry becomes the
    // first entry and its previous_entry link no longer points to an
    // entry.  If the list is empty, however, the list's tail link no
    // longer points to an entry.

    head = the_entry->Next;
    if (head != NULL)
      {
      head->Previous = NULL;
      }
    else
      {
            tail = NULL;
      }

    // The list's links are in place, free the removed entry and decrement
    // the counter.

    delete the_entry;
    --NumberEntries;
    }
  return(the_data);

}

template <class T> void TList<T>::AddEntry(T *new_data)
{
  TEntry<T>  *new_entry;
  TEntry<T>  *entry_ptr;  // pointer for traversing the list

  // Create a new entry and increment the counter

  new_entry = new TEntry<T>(new_data);
  ++NumberEntries;

  // If the list is empty, set the head and tail pointers to the new entry.

  if (head == NULL)
    {
    head = new_entry;
    tail = new_entry;
    }
  else
    {
    // Add the new entry to the back of the list

    entry_ptr = tail;

    // Set the new entry's pointers

    new_entry->Next = entry_ptr->Next;
    new_entry->Previous = entry_ptr;

    // Link the new entry

    entry_ptr->Next = new_entry;
    tail = new_entry;
    }

}

#endif

#include <windows.h>
#include "map_entity.h"

map_entity :: map_entity(unsigned entity_type, unsigned size)
```

```
{
unsigned i;

  type = entity_type;
  number_of_pts = size;

  if ((pt = new POINT[size]) == NULL)
    {
    MessageBox (NULL, "Unable to allocate pt array - out of memory", "",
            MB_OK | MB_ICONWARNING ) ;
    exit(-1);
    }

  for(i = 0; i < size; i++)
    pt[i].x = pt[i].y = 0;
}

map_entity :: ~map_entity(void)
{
  delete [] pt;
}

void map_entity :: set_xy_coord(unsigned index, long x, long y)
{
  pt[index].x = x;
  pt[index].y = y;
}

#include <windows.h>
#include "offsite.h"

template <class T> void TMapEntityList<T>::Draw(HDC hdc, BOOL show,
                                HPEN *hPen)
{
TEntry<T> *entry_ptr;
map_entity *me;
POINT *pt;

  if (!show) return;

  entry_ptr = head;

  while (entry_ptr != NULL)
    {
    me = entry_ptr->Data;
    pt = me->get_pt();
    SelectObject(hdc, hPen[PEN_BLACK]);
    Polyline(hdc, pt, me->get_number_of_pts());

          entry_ptr = entry_ptr->Next;
    }

}

#include <windows.h>
#include "offsite.h"

template <class T> void TPathList<T>::Draw(int start, int destin,
                              HDC hdc, BOOL show_names,
                              HPEN *hPen)
{
TEntry<T> *entry_ptr;
int index;
int node1, node2;
RECT rect;

  entry_ptr = head;

  if (start < destin)
    {
    index = destin - start;
    if (G_node_array[start].cost[index].is_drive_cost)
      {
      SelectObject(hdc, hPen[PEN_BLUE]);
      MoveToEx(hdc, (int) G_node_array[start].x, (int) G_node_array[start].y, NULL);
      LineTo(hdc, (int) G_node_array[destin].x, (int) G_node_array[destin].y);
      return;
      }
    }
    else
    {
    index = start - destin;
    if (G_node_array[destin].cost[index].is_drive_cost)
      {
      SelectObject(hdc, hPen[PEN_BLUE]);
      MoveToEx(hdc, (int) G_node_array[start].x, (int) G_node_array[start].y, NULL);
      LineTo(hdc, (int) G_node_array[destin].x, (int) G_node_array[destin].y);
      return;
      }
    }
  while (entry_ptr != NULL)
    {
    node1 = entry_ptr->Data->node1;
    node2 = entry_ptr->Data->node2;

    SelectObject(hdc, hPen[PEN_BLUE]);
    MoveToEx(hdc, (int) G_node_array[node1].x, (int) G_node_array[node1].y, NULL);
    LineTo(hdc, (int) G_node_array[node2].x, (int) G_node_array[node2].y);

    SelectObject(hdc, hPen[PEN_GREEN]);
    if ((node1 != start) && (node1 != destin))
      {
      rect.left = (long) G_node_array[node1].x - 20 ;
      rect.top = (long) G_node_array[node1].y + 20 ;
      rect.right = (long) G_node_array[node1].x + 20 ;
      rect.bottom = (long) G_node_array[node1].y - 20 ;

      Ellipse (hdc, rect.left, rect.top, rect.right, rect.bottom) ;

      if (show_names)
        {
        TextOut(hdc, rect.left, rect.top, G_node_array[node1].rics,
              strlen(G_node_array[node2].rics)) ;
        }
      }

    if ((node2 != start) && (node2 != destin))
      {
      rect.left = (long) G_node_array[node2].x - 20 ;
      rect.top = (long) G_node_array[node2].y + 20 ;
      rect.right = (long) G_node_array[node2].x + 20 ;
      rect.bottom = (long) G_node_array[node2].y - 20 ;

      Ellipse (hdc, rect.left, rect.top, rect.right, rect.bottom) ;

      if (show_names)
        {
        TextOut(hdc, rect.left, rect.top, G_node_array[node2].rics,
              strlen(G_node_array[node2].rics)) ;
        }
      }

          entry_ptr = entry_ptr->Next;
    }

}
```

```
#include <iostream.h>
#include <dir.h>
#include <stdio.h>
#include <stdlib.h>
#include "offsite.h"

void determine_travel_costs(void)
{
FILE *cost_fp;
int i, j;
int node_index;
char message[256];

  if ((cost_fp = fopen(G_cost_filename, "w")) == NULL)
    {
    sprintf (message, "Can't open output file %s", G_cost_filename) ;
    MessageBox (NULL, message, "", MB_OK | MB_ICONWARNING ) ;
    return;
    }

  G_number_sites = 0;

  for (i = 1; i <= G_number_nodes; i++)
    {
    G_best_sites[i].cost = -1;
    }

  for (i = 1; i <= G_number_nodes; i++)
    {
    if (G_node_array[i].is_site)
      {
      int cost = 0;
      fprintf(cost_fp,
            "Conference site: %s (%s, %s)\n\n",
            G_node_array[i].rics, G_node_array[i].city,
            state_code[G_node_array[i].state]);
      fprintf(cost_fp,
            "         # of     |                 Costs\n");
      fprintf(cost_fp,
            " Node | Participants |  Travel   |   Meals   |  Lodging  | Total\n");

      for (j = 1; j <= G_number_nodes; j++)
        {
        if (G_node_array[j].participants_solve)
          {
          cost += tdy_cost_to_site(i, j, cost_fp);
```

```
         }
       }
     fprintf(cost_fp,
          "                                    ----------\n");
     fprintf(cost_fp,
          " Total                                %10.2f\n\n",
          (float) cost);


     if (G_number_sites)
       {
       for (j = 1; j <= G_number_sites; j++)
         {
         if (cost < G_best_sites[j].cost)
           {
           for (int k = G_number_sites+1; k >= j+1; k--)
             {
             int prev_index = k-1;
             G_best_sites[k].site = G_best_sites[prev_index].site;
             G_best_sites[k].cost = G_best_sites[prev_index].cost;
             }

           G_best_sites[j].site = i;
           G_best_sites[j].cost = cost;

           break;
           }
         }

      if (j > G_number_sites)
        {
        G_best_sites[j].site = i;
        G_best_sites[j].cost = cost;
        }
      }
    else
      {
      G_best_sites[1].site = i;
      G_best_sites[1].cost = cost;
      }
    G_number_sites++;
    }
  }

if (G_number_sites > 1)
  {
    if (G_number_sites > 10)
      fprintf(cost_fp, " Ten Best Sites\n");
    else
      fprintf(cost_fp, " Cost to Sites\n");

    fprintf(cost_fp, " Site |  Cost\n");
    for (i = 1; i <= G_number_sites && i <= 10; i++)
      {
      node_index = G_best_sites[i].site;

      fprintf(cost_fp, " %4s | %10.2f\n",
           G_node_array[node_index].rics, (float) G_best_sites[i].cost);
      }
    }
  else
    {
    node_index = G_best_sites[G_number_sites].site;
    fprintf(cost_fp, " Cost to %s: %10.2f\n",
               G_node_array[node_index].rics,
               (float) G_best_sites[G_number_sites].cost);
    }
  fclose(cost_fp);
}

int tdy_cost_to_site(unsigned site, unsigned origin, FILE *cost_fp)
{
int tot_cost;
int travel_cost = 0;
int meals_travel;
int meals = 0;
int lodging = 0;
unsigned char cost_is_for_driving;

  fprintf(cost_fp, " %4s |   %4d     |",
      G_node_array[origin].rics, G_node_array[origin].participants_solve);

  if (site != origin)
    {
    int index;
    if (origin < site )
      {
      index = site - origin;
      travel_cost = G_node_array[origin].cost[index].amount;
      cost_is_for_driving = G_node_array[origin].cost[index].is_drive_cost;
      }
    else
    {
    index = origin - site;
    travel_cost = G_node_array[site].cost[index].amount;
    cost_is_for_driving = G_node_array[site].cost[index].is_drive_cost;
    }
  }

tot_cost = G_node_array[origin].participants_solve * 2 * travel_cost;

fprintf(cost_fp, " %10.2f |", (float) tot_cost);

if (site != origin)
  {
  if (cost_is_for_driving)
    {
    lodging = G_node_array[site].lodging *
           G_node_array[origin].participants_solve *
           (G_conf_duration - 1);

    if (G_conf_duration > 1)
      {
      meals_travel = (float) 0.75 * (float) G_node_array[site].meals *
                (float) G_node_array[origin].participants_solve *
                (float) 2.0;
      meals = meals_travel +
           (G_node_array[site].meals *
            G_node_array[origin].participants_solve *
            (G_conf_duration - 2));
      }
    else
      {
      meals_travel = (float) 0.75 * (float) G_node_array[site].meals *
                (float) G_node_array[origin].participants_solve;
      meals = meals_travel;
      }
    }
  else
    {
    lodging = G_node_array[site].lodging *
          G_node_array[origin].participants_solve * G_conf_duration;

    meals_travel = (float) 0.75 * (float) G_node_array[site].meals *
              (float) G_node_array[origin].participants_solve *
              (float) 2.0;

    if (G_conf_duration > 1)
```

```
        {
        meals = meals_travel +
              (G_node_array[site].meals *
               G_node_array[origin].participants_solve *
               (G_conf_duration - 1));
        }
      }
    }

  tot_cost += meals + lodging;

  fprintf(cost_fp, " %10.2f | %10.2f | %10.2f\n",
        (float) meals, (float) lodging, (float) tot_cost);

  return tot_cost;
}

#include <iostream.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "offsite.h"
#include "read.h"
#include "lat_long.h"
#include "constants.h"
#include "macros.h"

#define TOKEN_NOT_VALID 0
#define TOKEN_WEST_COAST_DATA 1
#define TOKEN_EAST_COAST_DATA 2
#define TOKEN_CONFERENCE_PARTICIPANT_DATA 3

static float min_lat, max_lat, min_long, max_long, center_lat, center_long;

/*
  Function:   is_comment

  Date:       January 1995

  Programmer:  David J. Ward

  Description:

     This function determines if the supplied text is considered
   a comment line. A comment line is an input line that is empty or
```

```
    begins with the '#' character.

  Modifications:


*/
int is_comment(char *token)
{
int index, length;

  /* If a null pointer is passed in return as if the line was a comment */
  if (token == NULL) return 1;

  /* Get the length of the text that was passed in to this function */
  length = (int) strlen(token);

  /* If this is a blank line */
  if (length == 0) return 1;

  /* starting at the begining of the text determine if this line is a comment */
  for (index = 0; index < length; index++)
    {

    /* if the character is white space continue to the next character */
    if (isspace(token[index])) continue;

    /* if we encounter a '#' then this line is a comment */
    if (token[index] == '#') return 1;

    /* if we reach the end of the text the line must have been blank */
    if (token[index] == '\0') return 1;

    /* If we have encontered a non-white space character and it wasn't
      a '#' or '\0', which was checked above, this line is not a
      comment line so we return 0. */
    return 0;
    }
}

/*
  Function:   is_token

  Date:       January 1995

  Programmer:  David J. Ward
  Description:

      This function determines if the first word of the input text is
    one of the recognized tokens. If it is the token value is returned.

  Modifications:

*/
int is_token(char *text)
{
char token[256];

  sscanf(text, "%s", token);

  if (strcmp(token, "WestCoast") == 0)
    return TOKEN_WEST_COAST_DATA;
  if (strcmp(token, "EastCoast") == 0)
    return TOKEN_EAST_COAST_DATA;
  if (strcmp(token, "NumberOfParticipants") == 0)
    return TOKEN_CONFERENCE_PARTICIPANT_DATA;
return TOKEN_NOT_VALID;
}

void parse_arc_line(int line_number, char *buf)
{
static int index = 0;
int start, end, cost;

  index++;

  if (sscanf(buf, "%d%d%d", &start, &end, &cost) != 3)
    {
    cout << "Unrecognizable arc data specified at line "
        << line_number << endl;
    return;
    }

  if (start < 1 || start > G_number_nodes)
    {
    cout << "Arc " << index << " has out of bounds start node ID "
        << start << " specified at line " << line_number << endl;
    return;
    }

  G_arc_array[index].start_node = start;
  if (end < 1 || end > G_number_nodes)
    {
    cout << "Arc " << index << " has out of bounds end node ID "
        << end << " specified at line " << line_number << endl;
    return;
    }

  G_arc_array[index].end_node = end;
  G_arc_array[index].cost = cost;

}

void parse_conference_duration_line(int line_number, char *buf)
{
  if (sscanf(buf, "%d", &G_conf_duration) != 1)
    cout << "Unrecognizable conference duration specified at line "
        << line_number << endl;
}

void parse_conference_site_line(int line_number, char *buf)
{
char airport_id[128];
int hash_val;
int node_id;

  if (sscanf(buf, "%s", &airport_id) != 1)
    {
    cout << "Unrecognizable airport ID specified at line"
        << line_number << endl;
    return;
    }

  if ((hash_val = G_airports->find_entry_in_lookup_table(airport_id)) == -1)
    {
    cout << "Unknown airport ID " << airport_id << " specified at line "
        << line_number << endl;
    return;
    }

  node_id = G_airports->get_node_id(hash_val);
  G_node_array[node_id].is_site = TRUE;
}

void parse_drive_cost_line(int line_number, char *buf)
{
int start, end;
```

```
 int drive_cost;
 float distance;
 int index;

   if (sscanf(buf, "%d%d%f%d",
           &start, &end, &distance, &drive_cost) != 4)
     {
     cout << "Unrecognizable cost data specified at line "
         << line_number << endl;
     return;
     }

   if (start < 1 || start > G_number_nodes)
     {
     cout << "Out of bounds start node ID " << start << " specified at line "
         << line_number << endl;
     return;
     }

   if (end < 1 || end > G_number_nodes)
     {
     cout << "Out of bounds end node ID " << start << " specified at line "
         << line_number << endl;
     return;
     }

   if (start < end)
     {
     index = end - start;
     G_node_array[start].cost[index].amount = drive_cost;
     G_node_array[start].cost[index].is_drive_cost = TRUE;
     }
   else
     {
     index = start - end;
     G_node_array[end].cost[index].amount = drive_cost;
     G_node_array[end].cost[index].is_drive_cost = TRUE;
     }
}

void parse_flight_cost_line(int line_number, char *buf)
{
int origin, destin;
float fcost;

  if (sscanf(buf, "%d%d%f", &origin, &destin, &fcost) != 3)
```

```
    {
    cout << "Unrecognizable cost data specified at line "
        << line_number << endl;
    return;
    }

  if (destin > origin)
    {
    int index = destin - origin;

    G_node_array[origin].cost[index].amount = (unsigned) fcost;
    }

}

void parse_node_line(int line_number, char *buf)
{
int node;
int lat_deg, lat_min, lat_sec, lat_dec_sec;
int long_deg, long_min, long_sec, long_dec_sec;
double x, y;

  if (sscanf(buf, "%d%d%d%d%d%d%d%d%d",
          &node, &lat_deg, &lat_min, &lat_sec, &lat_dec_sec,
          &long_deg, &long_min, &long_sec, &long_dec_sec) != 9)
    {
    cout << "Unrecognizable node data specified at line "
        << line_number << endl;
    return;
    }

  if (node < 1 || node > G_number_nodes)
    {
    cout << "Out of bounds node ID " << node << " specified at line "
          << line_number << endl;
    return;
    }

  G_node_array[node].x = lat_deg + (lat_min / MINUTES_PER_DEGREE) +
                          ((lat_sec + (ONE_THOUSANDTH * lat_dec_sec)) /
                           SECONDS_PER_DEGREE);
  G_node_array[node].y = long_deg + (long_min / MINUTES_PER_DEGREE) +
                          ((long_sec + (ONE_THOUSANDTH * long_dec_sec)) /
                           SECONDS_PER_DEGREE);

  lat_long_to_x_y(center_lat, center_long,
            G_node_array[node].x, G_node_array[node].y, &x, &y);
  G_node_array[node].x = x;
  G_node_array[node].y = y;

   //  min_lat = min(min_lat, G_node_array[node].x);
//  min_long = min(min_long, G_node_array[node].y);
//  max_lat = max(max_lat, G_node_array[node].x);
//  max_long = max(max_long, G_node_array[node].y);
}

void parse_per_diem_line(int line_number, char *buf)
{
int node;
int meals, lodging;
char *tokptr;
char state[3];
char city_name[12], short_name[10];
int hash_val, city_id;

 tokptr = strtok(buf, " ");

 node = atoi(tokptr);

 if (node < 1 || node > G_number_nodes)
   {
   cout << "Out of bounds node ID " << node << " specified at line "
        << line_number << endl;
   return;
   }

 skip_leading_whitespace(&tokptr);

 tokptr = strtok(NULL, " ");

 meals = atoi(tokptr);

 if (meals < 0)
   {
   cout << "Meal per diem for node " << node
      << " is not positive" << " at line " << line_number << endl;
   return;
   }

 G_node_array[node].meals = meals;

 skip_leading_whitespace(&tokptr);
```

```
tokptr = strtok(NULL, " ");

lodging = atoi(tokptr);

if (lodging < 0)
  {
  cout << "Lodging per diem for node " << node
      << " is not positive" << " at line " << line_number << endl;
  return;
  }

G_node_array[node].lodging = lodging;

skip_leading_whitespace(&tokptr);

tokptr = strtok(NULL, " ");

strncpy(G_node_array[node].rics, tokptr, 3);

G_airports->insert_entry_in_lookup_table(tokptr, node);

tokptr = strtok(NULL, "”");
tokptr = strtok(NULL, "”");

strncpy(G_node_array[node].city, tokptr, 15);

memset(short_name, '\0’, 10);
strncpy(short_name, G_node_array[node].city, 9);

skip_leading_whitespace(&tokptr);

tokptr = strtok(NULL, " \n");
strncpy(state, tokptr, 3);

if ((G_node_array[node].state = state_index(state)) == -1)
  {
  cout << "Unknown state abbreviation for node " << node
      << " at line " << line_number << endl;
  }

sprintf(city_name, "%s%s", short_name, state);
hash_val = G_cities->find_entry_in_lookup_table(city_name);

if (hash_val == -1)
  {
    city_id = ++G_number_cities;
    hash_val = G_cities->insert_entry_in_lookup_table(city_name);
    G_cities->set_city_id(hash_val, city_id);

    strcpy(G_city_array[city_id].city, G_node_array[node].city);

    if ((G_city_array[G_number_cities].airports = new TList<airport_record>) ==
NULL)
      {
      MessageBox (NULL, "Unable to allocate airport list - out of memory",
             "", MB_OK | MB_ICONWARNING ) ;
      exit(-1);
      }
    }
  else
    {
    city_id = G_cities->get_city_id(hash_val);
    }

  airport_record *new_airport = new airport_record;
  if (new_airport == NULL)
    {
    MessageBox (NULL, "Unable to allocate airport record - out of memory",
           "", MB_OK | MB_ICONWARNING ) ;
    exit(-1);
    }

  new_airport->id = node;
  G_city_array[city_id].airports->AddEntry(new_airport);
  G_city_array[city_id].state = G_node_array[node].state;
}

void read_arc_file(char *arc_filename)
{
FILE *arc_fp;
int i;
char message[256] ;
char input_line[1024];
char *bufptr;
int lineno = 0;

  if ((arc_fp = fopen(arc_filename, "r")) == NULL)
    {
    sprintf (message, "Can't open input file %s", arc_filename) ;
    MessageBox (NULL, message, "", MB_OK | MB_ICONWARNING ) ;
    exit(-1);
  }

fgets(input_line, sizeof(input_line), arc_fp);
sscanf(input_line, "%d %d", &G_number_nodes, &G_number_arcs);

G_arc_array = new arc[G_number_arcs+1];

if (G_arc_array == NULL)
  {
  MessageBox (NULL, "Unable to allocate G_arc_array - out of memory", "",
          MB_OK | MB_ICONWARNING ) ;
  exit(-1);
  }

G_node_array = new node[G_number_nodes+1];

if (G_node_array == NULL)
  {
  MessageBox (NULL, "Unable to allocate G_node_array - out of memory", "",
          MB_OK | MB_ICONWARNING ) ;
  exit(-1);
  }

for (i = 1; i <= G_number_nodes; i++)
  {
  G_node_array[i].rics[0] = '\0';
  G_node_array[i].city[0] = '\0';
  G_node_array[i].state = 0;
  G_node_array[i].participants_temp = 0;
  G_node_array[i].participants_solve = 0;
  G_node_array[i].meals = 0;
  G_node_array[i].lodging = 0;
  G_node_array[i].is_site = FALSE;
  G_node_array[i].cost = NULL;
  }

G_best_sites = new potential_site[G_number_nodes+1];

if (G_best_sites == NULL)
  {
  MessageBox (NULL, "Unable to allocate G_best_sites - out of memory", "",
          MB_OK | MB_ICONWARNING ) ;
  exit(-1);
  }

for (i = 1; i <= G_number_nodes; i++)
```

```
    {
    G_best_sites[i].site = 0;
    G_best_sites[i].cost = 0;
    }

  G_city_array = new city_and_airport[G_number_nodes+1];

  if (G_city_array == NULL)
    {
    MessageBox (NULL, "Unable to allocate G_city_array - out of memory", "",
            MB_OK | MB_ICONWARNING ) ;
    exit(-1);
    }

  for (i = 1; i <= G_number_nodes; i++)
    {
    G_city_array[i].city[0] = '\0';
    G_city_array[i].airports = NULL;
    }

  cout << "Reading " << arc_filename << endl;

  for(;;)
    {
    // Get the next input line
    fgets(input_line, sizeof(input_line), arc_fp);
    lineno++;

    // If we are at the end of file break, out of this loop
    if (feof(arc_fp)) break;

    // Assign bufptr the base address of the input line
    bufptr = input_line;

    // Make sure bufptr is pointing to the first non-whitespace
    // character in the input line.
    skip_leading_whitespace(&bufptr);

    // If this line is a comment line continue to the next line
    if (is_comment(bufptr)) continue;

    parse_arc_line(lineno, bufptr);
    }
  cout << "Done with " << arc_filename << endl;
  fclose(arc_fp);
}
```

```
void read_drive_cost(char *cost_filename)
{
FILE *cost_fp;
char message[256] ;
char input_line[1024];
char *bufptr;
int lineno = 0;

  if ((cost_fp = fopen(cost_filename, "r")) == NULL)
    {
    sprintf (message, "Can't open input file %s", cost_filename) ;
    MessageBox (NULL, message, "", MB_OK | MB_ICONWARNING ) ;
    exit(-1);
    }

  cout << "Reading " << cost_filename << endl;

  for(;;)
    {
    // Get the next input line
    fgets(input_line, sizeof(input_line), cost_fp);
    lineno++;

    // If we are at the end of file break, out of this loop
    if (feof(cost_fp)) break;

    // Assign bufptr the base address of the input line
    bufptr = input_line;

    // Make sure bufptr is pointing to the first non-whitespace
    // character in the input line.
    skip_leading_whitespace(&bufptr);

    // If this line is a comment line continue to the next line
    if (is_comment(bufptr)) continue;

    parse_drive_cost_line(lineno, bufptr);
    }
  cout << "Done with " << cost_filename << endl;
  fclose(cost_fp);
}

void read_flight_cost(char *cost_filename)
{
FILE *cost_fp;
```

```
int i;
char message[256] ;
char input_line[1024];
char *bufptr;
int lineno = 0;

  if ((cost_fp = fopen(cost_filename, "r")) == NULL)
    {
    sprintf (message, "Can’t open input file %s", cost_filename) ;
    MessageBox (NULL, message, "", MB_OK | MB_ICONWARNING ) ;
    exit(-1);
    }

  for (i = 1; i < G_number_nodes; i++)
    {
    int number_of_destins = G_number_nodes - i;

    G_node_array[i].cost = new travel_cost[number_of_destins+1];

    if  (G_node_array[i].cost == NULL)
      {
      MessageBox (NULL, "Unable to allocate cost array - out of memory", "",
              MB_OK | MB_ICONWARNING ) ;
      exit(-1);
      }

    for (int j = 1; j <= number_of_destins; j++)
      {
      G_node_array[i].cost[j].amount = 0;
      G_node_array[i].cost[j].is_drive_cost = FALSE;
      G_node_array[i].cost[j].path = new TPathList<transit_record>;

      if (G_node_array[i].cost[j].path == NULL)
        {
        MessageBox (NULL, "Unable to allocate path list - out of memory", "",
                MB_OK | MB_ICONWARNING ) ;
        exit(-1);
        }
      }
    }

 G_node_array[0].cost = G_node_array[G_number_nodes].cost = NULL;

 cout << "Reading " << cost_filename << endl;

 for(;;)
```

```
      {
      // Get the next input line
      fgets(input_line, sizeof(input_line), cost_fp);
      lineno++;

      // If we are at the end of file break, out of this loop
      if (feof(cost_fp)) break;

      // Assign bufptr the base address of the input line
      bufptr = input_line;

      // Make sure bufptr is pointing to the first non-whitespace
      // character in the input line.
      skip_leading_whitespace(&bufptr);

      // If this line is a comment line continue to the next line
      if (is_comment(bufptr)) continue;

      parse_flight_cost_line(lineno, bufptr);
      }
    cout << "Done with " << cost_filename << endl;
    fclose(cost_fp);

#ifdef ECHO
    for (i=1; i <= G_number_nodes; i++)
      {
      int number_of_destins = G_number_nodes - i;
      int j;

      for (j=1; j <= number_of_destins; j++)
        cout << i << " " << i+j << " " << G_node_array[i].cost[j].amount << endl;
      }
#endif
}

void read_node_file(char *node_filename)
{
FILE *node_fp;
//int i;
char message[256] ;
char input_line[1024];
char *bufptr;
int lineno = 0;
//float center_lat, center_long;
//double x, y;
```

```
   if ((node_fp = fopen(node_filename, "r")) == NULL)
     {
     sprintf (message, "Can't open input file %s", node_filename) ;
     MessageBox (NULL, message, "", MB_OK | MB_ICONWARNING ) ;
     exit(-1);
     }

//  min_lat = 90.0;
//  min_long = 180.0;
//  max_lat = -90.0;
//  max_long = -180.0;

   cout << "Reading " << node_filename << endl;

   for(;;)
     {
     // Get the next input line
     fgets(input_line, sizeof(input_line), node_fp);
     lineno++;

     // If we are at the end of file break, out of this loop
     if (feof(node_fp)) break;

     // Assign bufptr the base address of the input line
     bufptr = input_line;

     // Make sure bufptr is pointing to the first non-whitespace
     // character in the input line.
     skip_leading_whitespace(&bufptr);

     // If this line is a comment line continue to the next line
     if (is_comment(bufptr)) continue;

     parse_node_line(lineno, bufptr);
     }
  cout << "Done with " << node_filename << endl;
  fclose(node_fp);

//  lat_long_to_x_y(min_lat, min_long, max_lat, max_long,
//              &x, &y);

//  G_map_width = x;
//  G_map_height = y;
//  center_lat = (min_lat + max_lat) / 2.0;
//  center_long = (min_long + max_long) / 2.0;
//  for (i = 1; i <= G_number_nodes; i++)
//    {
//    lat_long_to_x_y(center_lat, center_long,
//                  G_node_array[i].x, G_node_array[i].y, &x, &y);
//    G_node_array[i].x = x;
//    G_node_array[i].y = y;
//    }
}

void read_per_diem_file(char *per_diem_filename)
{
FILE *per_diem_fp;
char message[256] ;
char input_line[1024];
char *bufptr;
int lineno = 0;

  if ((per_diem_fp = fopen(per_diem_filename, "r")) == NULL)
    {
    sprintf (message, "Can't open input file %s", per_diem_filename) ;
    MessageBox (NULL, message, "", MB_OK | MB_ICONWARNING ) ;
    exit(-1);
    }

  cout << "Reading " << per_diem_filename << endl;

  for(;;)
    {
    // Get the next input line
    fgets(input_line, sizeof(input_line), per_diem_fp);
    lineno++;

    // If we are at the end of file break, out of this loop
    if (feof(per_diem_fp)) break;

    // Assign bufptr the base address of the input line
    bufptr = input_line;

    // Make sure bufptr is pointing to the first non-whitespace
    // character in the input line.
    skip_leading_whitespace(&bufptr);

    // If this line is a comment line continue to the next line
    if (is_comment(bufptr)) continue;

    parse_per_diem_line(lineno, bufptr);
    }
  cout << "Done with " << per_diem_filename << endl;
  fclose(per_diem_fp);

#ifdef ECHO
  int i;
  for (i=1; i <= G_number_nodes; i++)
    {
    cout << "Per diem at node " << i << " is "
       << G_node_array[i].meals << " (meals) "
       << G_node_array[i].lodging << " (lodging)" << endl;
    }
#endif
}

void read_route_file(char *route_filename)
{
FILE *route_fp;
char message[256] ;
char input_line[1024];
char label[15];
char arc_label[15];
int start_node, destin_node;
int arc_id, from_node, to_node, index;
long lineno = 0;

  if ((route_fp = fopen(route_filename, "r")) == NULL)
    {
    sprintf (message, "Can’t open input file %s", route_filename) ;
    MessageBox (NULL, message, "", MB_OK | MB_ICONWARNING ) ;
    exit(-1);
    }

  cout << "Reading " << route_filename << endl;

  while (!feof(route_fp))
    {
    fgets(input_line, sizeof(input_line), route_fp);
    lineno++;

    if (feof(route_fp)) break;

    if (sscanf(input_line, "%s%s%d%s%s%d",
                  label, label, &start_node,
                  label, label, &destin_node) != 6)
     {
```

```
  sprintf (message, "Unrecognizable node data specified at line %d",
        lineno);
  MessageBox (NULL, message, route_filename, MB_OK | MB_ICONWARNING );
  fclose(route_fp);
  return;
  }

if (start_node > destin_node)
  {
  for (;;)
    {
    fgets(input_line, sizeof(input_line), route_fp);
    lineno++;

    if (feof(route_fp)) break;

    if (isspace(input_line[0]))
      {
      fgets(input_line, sizeof(input_line), route_fp);
      lineno++;
      break;
      }
    }

  continue;
  }

for (;;)
  {
  fgets(input_line, sizeof(input_line), route_fp);
  lineno++;

  if (feof(route_fp)) break;

  if (isspace(input_line[0]))
    {
    fgets(input_line, sizeof(input_line), route_fp);
    lineno++;
    break;
    }

  if (sscanf(input_line, "%s%d%s%s%d%s%s%d",
                  label, &arc_id,
                  label, label, &from_node,
                  label, label, &to_node) != 8)
    {
            sprintf (message, "Unrecognizable arc data specified at line %d",
                  lineno);
            MessageBox (NULL, message, route_filename, MB_OK | MB_ICONWARNING
   );
            fclose(route_fp);
            return;
            }

         transit_record *new_transit = new transit_record;
         if (new_transit == NULL)
           {
           MessageBox (NULL, "Unable to allocate transit record - out of memory",
                   "", MB_OK | MB_ICONWARNING ) ;
           exit(-1);
           }

         new_transit->node1 = from_node;
         new_transit->node2 = to_node;

         if (start_node < destin_node)
           {
           index = destin_node - start_node;
           G_node_array[start_node].cost[index].path->AddEntry(new_transit);
           }
         else
           {
           index = start_node - destin_node;
           G_node_array[destin_node].cost[index].path->AddEntry(new_transit);
           }

         } // loop and read the next arc
       } // loop and read the next line

    cout << "Done with " << route_filename << endl;
    fclose(route_fp);
  }

  void read_usa_map(char *map_filename)
  {
  FILE *map_fp;
  char message[256] ;
  char input_line[1024];
  int i;
  int type, size;
  float latitude, longitude;
  double x, y;
```

```
map_entity *me;

 if ((G_map_entities = new TMapEntityList<map_entity>) == NULL)
   {
   MessageBox (NULL, "Unable to allocate graphic entities list - out of memory",
          "", MB_OK | MB_ICONWARNING ) ;
   exit(-1);
   }

 if ((map_fp = fopen(map_filename, "r")) == NULL)
   {
   sprintf (message, "Can't open input file %s", map_filename) ;
   MessageBox (NULL, message, "", MB_OK | MB_ICONWARNING ) ;
   exit(-1);
   }

 min_lat = 90.0;
 min_long = 180.0;
 max_lat = -90.0;
 max_long = -180.0;

 cout << "Reading " << map_filename << endl;

 fgets(input_line, sizeof(input_line), map_fp);

 if (sscanf(input_line, "%f%f", &center_lat, &center_long) != 2)
   {
   MessageBox (NULL, "Unrecognizable center lat/long specified", "",
          MB_OK | MB_ICONWARNING ) ;
   exit(-1);
   }

 center_long = fabs(center_long);

 for(;;)
   {
   fgets(input_line, sizeof(input_line), map_fp);

   if (feof(map_fp)) break;

   if (sscanf(input_line, "%d%d", &type, &size) != 2)
     {
     MessageBox (NULL, "Unrecognizable coordinate block type and size specified",
"",
            MB_OK | MB_ICONWARNING ) ;
     exit(-1);
```

```
      }

    if ((me = new map_entity(type, size)) == NULL)
      {
      MessageBox (NULL, "Unable to allocate graphic entity - out of memory", "",
              MB_OK | MB_ICONWARNING ) ;
      exit(-1);
      }

    for (i = 0; i < size; i++)
      {
      fscanf(map_fp, "%f%f", &latitude, &longitude);
      longitude = fabs(longitude);

      min_lat = min(min_lat, latitude);
      min_long = min(min_long, longitude);
      max_lat = max(max_lat, latitude);
      max_long = max(max_long, longitude);

      lat_long_to_x_y(center_lat, center_long, latitude, longitude, &x, &y);
      me->set_xy_coord(i, (long) x, (long) y);
      }

    G_map_entities->AddEntry(me);

    fgets(input_line, sizeof(input_line), map_fp);
    if (feof(map_fp)) break;
    }
  cout << "Done with " << map_filename << endl;
  fclose(map_fp);

  lat_long_to_x_y(min_lat, min_long, max_lat, max_long,
            &x, &y);

  G_map_width = x;
  G_map_height = y;
}

/*
  Function:    skip_leading_whitespace

  Date:        January 1995

  Programmer:  David J. Ward

  Description:
      This function skips the leading white space in the
    text pointed to by *tokenptr.

  Modifications:


*/
void skip_leading_whitespace(char **tokenptr)
{
int index, length;
char *token = *tokenptr;

  /* If a null pointer is passed in return */
  if (token == NULL) return;

  /* Get the length of the text that was passed in to this function */
  length = (int) strlen(token);

  /* starting at the begining of the text move forward until a non-white
    space character is found */
  for (index = 0; index < length; index++)
    {

    /* if the character is white space continue to the next character */
    if (isspace(token[index])) continue;

    break;
    }

  /* Change the value of the pointer pointed to by tokenptr so
    that it points to the first non-white space character.  */
  *tokenptr = &token[index];
}

int state_index(char *state)
{
int i;

  for (i = 0; i < 52; i++)
    {
    if (strcmp(state, state_code[i]) == 0) return i;
    }

  return -1;
}
```

```
#ifndef __constants_h__
#define __constants_h__

#include <math.h>

#define MINUTES_PER_DEGREE 60.0
#define SECONDS_PER_DEGREE 3600.0

#define DEG_PER_RAD 57.29577951  // degrees per radian
#define NM_PER_DEG  60.0         // nautical miles per degree
#define KM_PER_NM   1.853        // Kilometers per nautical mile
#define NM_PER_KM   0.5396
#define TWO_PI      2.0 * M_PI

#define ONE_THOUSANDTH 0.001

#endif

#ifndef __conf_map_entity_h__
#define __conf_map_entity_h__

class map_entity
{
private:
  unsigned type;
  unsigned number_of_pts;
  POINT *pt;

public:
  map_entity(unsigned entity_type, unsigned size);
  ~map_entity();
  unsigned get_number_of_pts(void) { return number_of_pts; } ;
  POINT *get_pt(void) { return pt; } ;
  void set_xy_coord(unsigned index, long x, long y);
};

#endif

#ifndef __read_h__
#define __read_h__

int is_comment(char *token);
int is_token(char *text);
void parse_arc_line(int line_number, char *buf);
void parse_conference_duration_line(int line_number, char *buf);
void parse_conference_site_line(int line_number, char *buf);
void parse_drive_cost_line(int line_number, char *buf);
void parse_flight_cost_line(int line_number, char *buf);
void parse_node_line(int line_number, char *buf);
void parse_per_diem_line(int line_number, char *buf);
void read_drive_cost(char *cost_filename);
void read_flight_cost(char *cost_filename);
void read_arc_file(char *arc_filename);
void read_node_file(char *node_filename);
void read_per_diem_file(char *per_diem_filename);
void read_route_file(char *route_filename);
void read_usa_map(char *map_filename);
void skip_leading_whitespace(char **tokenptr);
int state_index(char *state);

#endif

/*---------------------------------------------------------
  file_ops.cpp
 ---------------------------------------------------------*/

#include <windows.h>
#include "offsite.h"

void init_openfilename_struc (HWND hwnd, OPENFILENAME *ofn)
{
static char szFilter[]= "Text Files (*.TXT)\0*.txt\0" \
                    "All Files (*.*)\0*.*\0\0" ;
static char szInitialDir[] = "..\\out" ;

  ofn->lStructSize = sizeof (OPENFILENAME) ;
  ofn->hwndOwner = hwnd ;
  ofn->hInstance = NULL ;
  ofn->lpstrFilter = szFilter ;
  ofn->lpstrCustomFilter = NULL ;
  ofn->nMaxCustFilter = 0 ;
  ofn->nFilterIndex = 0 ;
  ofn->lpstrFile = NULL ;
  ofn->nMaxFile = _MAX_PATH ;
  ofn->lpstrFileTitle = NULL ;
  ofn->nMaxFileTitle = _MAX_FNAME + _MAX_EXT ;
  ofn->lpstrInitialDir = szInitialDir ;
  ofn->lpstrTitle = NULL ;
  ofn->Flags = 0 ;
  ofn->nFileOffset = 0 ;
  ofn->nFileExtension = 0 ;
  ofn->lpstrDefExt = "txt" ;
```

```
   ofn->lCustData = 0L ;
   ofn->lpfnHook = NULL ;
   ofn->lpTemplateName = NULL ;
  }

 void save_as_dlg(HWND hwnd, OPENFILENAME *ofn)
 {
 char buffer[256];
 char szFile[] = "noname.txt" ;
 FILE *read_fp, *write_fp;

   ofn->hwndOwner = hwnd ;
   ofn->lpstrFile = szFile ;
   ofn->Flags = OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT ;

   if (GetSaveFileName(ofn))
     {
    if ((read_fp = fopen(G_cost_filename, "r")) == NULL ||
        (write_fp = fopen(ofn->lpstrFile, "w")) == NULL)
       {
       sprintf (buffer, "Can't save %s", ofn->lpstrFile) ;
       MessageBox (hwnd, buffer, "", MB_OK | MB_ICONWARNING ) ;
       return ;
       }

     while(fgets(buffer, sizeof(buffer), read_fp) != NULL)
       {
       fprintf(write_fp, "%s", buffer);
       }

     fclose (read_fp) ;
     fclose (write_fp) ;
     }

}

#include "offsresource.h"

IDD_SolveDialog DIALOG 5, 1, 349, 298
STYLE DS_MODALFRAME | DS_3DLOOK | WS_POPUP | WS_VISIBLE |
WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX
CAPTION ""
FONT 8, "MS Sans Serif"
{
 CONTROL "&Cancel", IDCANCEL, "BUTTON", BS_PUSHBUTTON | BS_CENTER
| WS_CHILD | WS_VISIBLE | WS_TABSTOP, 179, 276, 50, 14
```

```
 CONTROL "Conference duration (days):", -1, "static", SS_LEFT | WS_CHILD |
WS_VISIBLE, 8, 9, 88, 8
 CONTROL "", IDC_Duration, "edit", ES_LEFT | ES_NUMBER | WS_CHILD |
WS_VISIBLE | WS_BORDER | WS_TABSTOP, 100, 6, 32, 12
 CONTROL "Departing city:", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 20,
43, 52, 8
 CONTROL "", IDC_DepartingCity, "listbox", LBS_STANDARD |
LBS_MULTIPLESEL | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 16, 52, 104, 60
 CONTROL "Departing airport:", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE,
136, 43, 60, 8
 CONTROL "", IDC_DepartingAirport, "combobox", CBS_DROPDOWNLIST |
CBS_SORT | CBS_NOINTEGRALHEIGHT | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 140, 52, 56, 56
 CONTROL "Number of participants:", -1, "static", SS_LEFT | WS_CHILD |
WS_VISIBLE, 208, 56, 76, 8
 CONTROL "", IDC_Participants, "edit", ES_LEFT | ES_NUMBER | WS_CHILD |
WS_VISIBLE | WS_BORDER | WS_TABSTOP, 288, 53, 32, 12
 CONTROL "Solve:", IDC_GROUPBOX1, "button", BS_GROUPBOX | WS_CHILD |
WS_VISIBLE | WS_GROUP, 8, 125, 324, 142
 CONTROL "Restrict conference sites to departure points", IDC_Case1, "button",
BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 32, 138,
156, 12
 CONTROL "Case 2", IDC_Case2, "button", BS_AUTORADIOBUTTON | WS_CHILD
| WS_VISIBLE | WS_TABSTOP, 32, 159, 112, 12
 CONTROL "Specify site", IDC_Case3, "button", BS_AUTORADIOBUTTON |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 32, 181, 52, 12
 CONTROL "Arriving city:", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 104,
183, 40, 8
 CONTROL "", IDC_ArrivingCity, "listbox", LBS_STANDARD | WS_CHILD |
WS_VISIBLE | WS_DISABLED | WS_TABSTOP, 104, 194, 104, 60
 CONTROL "Arriving airport:", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE,
232, 183, 52, 8
 CONTROL "", IDC_ArrivingAirport, "combobox", CBS_DROPDOWNLIST |
CBS_SORT | CBS_NOINTEGRALHEIGHT | WS_CHILD | WS_VISIBLE |
WS_DISABLED | WS_TABSTOP, 232, 194, 56, 56
 CONTROL "E&xecute", IDC_Execute, "button", BS_DEFPUSHBUTTON |
BS_CENTER | WS_CHILD | WS_VISIBLE | WS_DISABLED | WS_TABSTOP, 119,
276, 50, 14
 CONTROL "Departure point(s):", IDC_GROUPBOX2, "button", BS_GROUPBOX |
WS_CHILD | WS_VISIBLE | WS_GROUP, 8, 26, 324, 91
}

IDD_SummaryDialog DIALOG 5, 1, 276, 294
STYLE DS_MODALFRAME | DS_3DLOOK | WS_POPUP | WS_VISIBLE |
WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX
CAPTION "Summary"
```

```
FONT 8, "Fixedsys"
{
 CONTROL "&OK", IDOK, "BUTTON", BS_PUSHBUTTON | BS_CENTER |
WS_CHILD | WS_VISIBLE | WS_TABSTOP, 145, 260, 46, 24
 CONTROL "", IDC_DurationSummary, "static", SS_LEFT | WS_CHILD |
WS_VISIBLE, 8, 9, 100, 8
 CONTROL "Departure point(s):", IDC_GROUPBOX1, "button", BS_GROUPBOX |
WS_CHILD | WS_VISIBLE | WS_GROUP, 8, 26, 260, 91
 CONTROL "City", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 20, 43, 20, 8
 CONTROL "Airport", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE, 148, 43, 28,
8
 CONTROL "# of participants", -1, "static", SS_LEFT | WS_CHILD | WS_VISIBLE,
188, 43, 68, 8
 CONTROL "", IDC_DeparturePoint, "listbox", LBS_STANDARD |
LBS_USETABSTOPS | LBS_NOSEL | WS_CHILD | WS_VISIBLE | WS_TABSTOP,
20, 52, 236, 60
 CONTROL "", IDC_SingleSiteCost, "static", SS_LEFT | SS_SUNKEN | WS_CHILD |
NOT WS_VISIBLE, 8, 124, 260, 10
 CONTROL "", IDC_CostGroupBox, "button", BS_GROUPBOX | WS_CHILD | NOT
WS_VISIBLE | WS_GROUP, 8, 125, 260, 125
 CONTROL "", IDC_TotalCost, "listbox", LBS_NOTIFY | LBS_USETABSTOPS |
LBS_NOSEL | WS_CHILD | NOT WS_VISIBLE | WS_BORDER | WS_TABSTOP, 20,
153, 232, 88
 CONTROL "&Detailed Costs", IDC_DetailedCosts, "button", BS_DEFPUSHBUTTON
| BS_CENTER | BS_VCENTER | BS_MULTILINE | WS_CHILD | WS_VISIBLE |
WS_TABSTOP, 85, 260, 46, 24
 CONTROL "City", IDC_SiteStaticText, "static", SS_LEFT | WS_CHILD | NOT
WS_VISIBLE, 24, 145, 32, 8
 CONTROL "Airport", IDC_AirportStaticText, "static", SS_LEFT | WS_CHILD | NOT
WS_VISIBLE, 148, 145, 28, 8
 CONTROL "Total cost", IDC_CostStaticText, "static", SS_LEFT | WS_CHILD | NOT
WS_VISIBLE, 200, 145, 40, 8
}

IDM_MainMenu MENU
{
 POPUP "&File"
 {
  MENUITEM "&Save As...", IDM_SaveAs, GRAYED
  MENUITEM SEPARATOR
  MENUITEM "E&xit", IDM_Exit
 }

 POPUP "&Solve"
 {
  MENUITEM "&Run...", IDM_Run
  }

 POPUP "S&how"
  {
   MENUITEM "&Airport Names", IDM_AirportNames
   MENUITEM "&Map", IDM_Map
  }

}

ICO_Network ICON "network.ico"

#ifndef __OFFSRESOURCE__
#define __OFFSRESOURCE__

#define IDM_MainMenu          100
#define IDM_SaveAs           101
#define IDM_Exit            102
#define IDM_Run             103
#define IDM_AirportNames       104
#define IDM_Map              105

#define IDD_SolveDialog        200
#define IDC_Duration          201
#define IDC_GROUPBOX1          202
#define IDC_DepartingCity      203
#define IDC_DepartingAirport   204
#define IDC_Participants       205
#define IDC_GROUPBOX2          206
#define IDC_Case1             207
#define IDC_Case2             208
#define IDC_Case3             209
#define IDC_ArrivingCity       210
#define IDC_ArrivingAirport    211
#define IDC_Execute           212

#define IDD_SummaryDialog      300
#define IDC_DurationSummary    301
#define IDC_DeparturePoint     302
#define IDC_SingleSiteCost     303
#define IDC_CostGroupBox       304
#define IDC_SiteStaticText     305
#define IDC_AirportStaticText  306
#define IDC_CostStaticText     307
#define IDC_TotalCost         308
#define IDC_DetailedCosts      309
```

```
#define ICO_Network          400

#endif /* __OFFSRESOURCE__ */

/*------------------------------------------------------
  summary_dlg_proc.cpp
 ------------------------------------------------------*/

#include <windows.h>
#include "offsite.h"
#include "offsresource.h"

BOOL CALLBACK summary_dlg_proc (HWND hDlg, UINT iMsg, WPARAM
wParam, LPARAM lParam)
{
int i;
char buffer[256];

  switch (iMsg)
    {
    case WM_INITDIALOG :
       {
       char temp[256];
       int node_index;

       // Display the conference duration in the duration static field
       HWND hwndStaticTxt = GetDlgItem(hDlg, IDC_DurationSummary) ;

       sprintf(buffer, "Conference duration: %d days", G_conf_duration) ;
       SetWindowText(hwndStaticTxt, buffer) ;

       HWND hwndList1 = GetDlgItem(hDlg, IDC_DeparturePoint) ;
       SendMessage(hwndList1, WM_SETREDRAW, FALSE, 0) ;

       for (i = 1; i <= G_number_nodes; i++)
         {
         if (G_node_array[i].participants_solve)
           {
           sprintf(temp, "%s, %s",
                    G_node_array[i].city,
                    state_code[G_node_array[i].state]) ;
           sprintf(buffer, "%-20s\t\t%s\t\t%d",
                     temp,
                     G_node_array[i].rics,
                     G_node_array[i].participants_solve) ;
```

```
            SendMessage(hwndList1, LB_ADDSTRING, 0, (LPARAM) buffer) ;
            }
          }

        SendMessage(hwndList1, WM_SETREDRAW, TRUE, 0) ;

        if (G_number_sites > 1)
          {
          HWND hwndGroupBox = GetDlgItem(hDlg, IDC_CostGroupBox) ;
          ShowWindow(hwndGroupBox, SW_SHOWNORMAL) ;
          if ( G_number_sites > 10)
            strcpy(buffer, "Ten Best Sites") ;
          else
            strcpy(buffer, "Cost to Sites") ;

          SetWindowText(hwndGroupBox, buffer) ;

          HWND hwndList2 = GetDlgItem(hDlg, IDC_TotalCost) ;
          ShowWindow(hwndList2, SW_SHOWNORMAL) ;
          SendMessage(hwndList2, WM_SETREDRAW, FALSE, 0) ;

          for (i = 1; i <= G_number_sites && i <= 10; i++)
            {
            node_index = G_best_sites[i].site;

            sprintf(temp, "%s, %s",
                      G_node_array[node_index].city,
                      state_code[G_node_array[node_index].state]) ;
            sprintf(buffer, "%-20s\t\t%s\t\t%-10.2f",
                       temp,
                       G_node_array[node_index].rics,
                       (float) G_best_sites[i].cost) ;

            SendMessage(hwndList2, LB_ADDSTRING, 0, (LPARAM) buffer) ;
            }

          SendMessage(hwndList2, WM_SETREDRAW, TRUE, 0) ;

          ShowWindow(GetDlgItem(hDlg, IDC_SiteStaticText), SW_SHOWNORMAL)
;
          ShowWindow(GetDlgItem(hDlg, IDC_AirportStaticText),
SW_SHOWNORMAL) ;
          ShowWindow(GetDlgItem(hDlg, IDC_CostStaticText), SW_SHOWNORMAL)
;
          }
        else
```

```
    {
    node_index = G_best_sites[G_number_sites].site;
    sprintf(buffer, " Cost to %s (%s, %s): %10.2f\n",
              G_node_array[node_index].rics,
              G_node_array[node_index].city,
              state_code[G_node_array[node_index].state],
              (float) G_best_sites[G_number_sites].cost);
    hwndStaticTxt = GetDlgItem(hDlg, IDC_SingleSiteCost) ;
    ShowWindow(hwndStaticTxt, SW_SHOWNORMAL) ;
    SetWindowText(hwndStaticTxt, buffer) ;
    }

  return TRUE ;
  }

case WM_COMMAND :
  switch (LOWORD (wParam))
    {
    case IDCANCEL :
    case IDOK :
      {
      HWND hwndButton = GetDlgItem(GetParent(hDlg), IDC_Execute) ;
      EnableWindow(hwndButton, TRUE) ;
      DestroyWindow(hDlg) ;
      return TRUE ;
      }

    case IDC_DetailedCosts :
      {
      char command_line[256];
      STARTUPINFO si;
      PROCESS_INFORMATION     pi;

      sprintf (command_line, "notepad.exe %s", G_cost_filename) ;
      memset((char *)&si, '\0', sizeof(si)) ;
      si.cb = sizeof(si) ;
      si.wShowWindow = SW_SHOWNORMAL ;
      if (!CreateProcess(NULL,
              command_line,
              NULL,
              NULL,
              FALSE,
              0,
              NULL,
              NULL,
              &si,
                        &pi))
                  {
                  sprintf (buffer, "Unable to execute\n%s", command_line) ;
                  MessageBox (NULL, buffer, "", MB_OK | MB_ICONWARNING ) ;
                  return TRUE ;
                  }
                CloseHandle(pi.hProcess) ;
                CloseHandle(pi.hThread) ;
                return TRUE ;
                }
            }
      }

   return FALSE ;

} // end of summary_dlg_proc

/*------------------------------------------------------------
   solv_dlg_proc.cpp
  ------------------------------------------------------------*/

#include <windows.h>
#include "offsite.h"
#include "offsresource.h"
#include "read.h"

BOOL CALLBACK solve_dlg_proc (HWND hDlg, UINT iMsg, WPARAM wParam,
LPARAM lParam)
{
int i ;
char buffer[80] ;
HWND hwndList1, hwndList2, hwndButton ;

  switch (iMsg)
    {
    case WM_INITDIALOG :
        {
        // Display the conference duration in the duration edit control
        HWND hwndEdit = GetDlgItem(hDlg, IDC_Duration) ;

        sprintf(buffer, "%d", G_conf_duration) ;
        SetWindowText(hwndEdit, buffer) ;
        }

        // Cities will be displayed in arriving and departure list boxes.
        // Don't allow either list box to update its window until they
// are filled.

hwndList1 = GetDlgItem(hDlg, IDC_DepartingCity) ;
SendMessage(hwndList1, WM_SETREDRAW, FALSE, 0) ;

hwndList2 = GetDlgItem(hDlg, IDC_ArrivingCity) ;
SendMessage(hwndList2, WM_SETREDRAW, FALSE, 0) ;

for (i = 1; i <= G_number_cities; i++)
  {
  sprintf(buffer, "%s, %s", G_city_array[i].city,
                    state_code[G_city_array[i].state]) ;
  SendMessage(hwndList1, LB_ADDSTRING, 0, (LPARAM) buffer) ;
  SendMessage(hwndList2, LB_ADDSTRING, 0, (LPARAM) buffer) ;
  }

// Finished filling the list boxes, turn the redraw flag back on

SendMessage(hwndList1, WM_SETREDRAW, TRUE, 0) ;
SendMessage(hwndList2, WM_SETREDRAW, TRUE, 0) ;

if (G_city_is_selected == NULL)
  {
  if ((G_city_is_selected = new BOOL[G_number_cities]) == NULL)
    {
    MessageBox (NULL, "Unable to allocate city array - out of memory", "",
            MB_OK | MB_ICONWARNING ) ;
    exit(-1);
    }

  for (i = 0; i < G_number_cities; i++)
    G_city_is_selected[i] = FALSE ;
  }

for (i = 0; i < G_number_cities; i++)
  {
  if (G_city_is_selected[i])
    SendMessage(hwndList1, LB_SETSEL, TRUE, i) ;
  }

// Place a label next to the case 2 radio button

hwndButton = GetDlgItem(hDlg, IDC_Case2) ;
sprintf(buffer, "Best cost over %d sites", G_number_nodes) ;
SetWindowText(hwndButton, buffer);
```

```
    return TRUE ;

case WM_COMMAND :
    switch (LOWORD (wParam))
      {
      case IDCANCEL :
          {
          HWND hwndEdit = GetDlgItem(hDlg, IDC_Participants) ;
          HWND hwndCombo = GetDlgItem(hDlg, IDC_DepartingAirport) ;
          save_number_of_participants(hwndEdit, hwndCombo) ;
          EnableMenuItem(GetMenu(GetParent(hDlg)), IDM_Run, MF_ENABLED) ;
          DestroyWindow(hDlg) ;
          return TRUE;
          }

      case IDC_Case1 :
          disable_Arriving_controls(hDlg) ;
          return TRUE;

      case IDC_Case2 :
          disable_Arriving_controls(hDlg) ;
          return TRUE;

      case IDC_Case3 :
          handle_Case3_radiobutton(hDlg) ;
          return TRUE;

      case IDC_Execute :
          handle_Execute_button(hDlg) ;
          return TRUE;
      }
   switch (HIWORD (wParam))
      {
      case LBN_SELCHANGE :       // == CBN_SELCHANGE

          switch (LOWORD (wParam))
            {
            case IDC_DepartingCity :
                handle_DepartingCity_listbox(hDlg, lParam) ;
                return TRUE;

            case IDC_ArrivingCity :
                handle_ArrivingCity_listbox(hDlg, lParam) ;
                return TRUE;

            case IDC_DepartingAirport :
                handle_DepartingAirport_combobox(hDlg, lParam) ;
                return TRUE;

            case IDC_ArrivingAirport :
                // Do nothing
                return TRUE;
             }

        }
    }

  return FALSE ;

} // end of solve_dlg_proc

void disable_Arriving_controls(HWND hDlg)
{
HWND hwndList = GetDlgItem(hDlg, IDC_ArrivingCity);
HWND hwndCombo = GetDlgItem(hDlg, IDC_ArrivingAirport);
HWND hwndButton = GetDlgItem(hDlg, IDC_Execute);

  EnableWindow(hwndList, FALSE) ;
  EnableWindow(hwndCombo, FALSE) ;
  EnableWindow(hwndButton, TRUE) ;

} // end of disable_Arriving_controls

void handle_ArrivingCity_listbox(HWND hDlg, LPARAM lParam)
{
HWND hwndList = (HWND) lParam ;
HWND hwndCombo = GetDlgItem(hDlg, IDC_ArrivingAirport) ;
int index = SendMessage(hwndList, LB_GETCURSEL, 0, 0) ;
int j;
int hash_val, city_id;
int node_id;
char buffer[80] ;
char *tokptr;
char city_name[12], short_name[10];
airport_record *data;

  // Clear out the arriving airport combo box
  SendMessage(hwndCombo, CB_RESETCONTENT, 0, 0) ;

  // Get the city and state from the list box's current selection
  SendMessage(hwndList, LB_GETTEXT, index, (LPARAM) buffer) ;
  // Extract the city name
  tokptr = strtok(buffer, ",") ;

  // Truncate the city name to first nine charaters
  memset(short_name, '\0', 10) ;
  strncpy(short_name, tokptr, 9) ;

  // Go to the two-letter state abbreviation
  tokptr = strtok(NULL, " ") ;
  skip_leading_whitespace(&tokptr);

  // Append state to city string for hash table search
  sprintf(city_name, "%s%2s", short_name, tokptr) ;

  // Obtain the city index
  hash_val = G_cities->find_entry_in_lookup_table(city_name) ;
  city_id = G_cities->get_city_id(hash_val) ;

  // Fill out the arriving airport combo box with the airports located
  // at this city
  for (j = 1; j <= G_city_array[city_id].airports->GetNumberEntries(); j++)
    {
    data = G_city_array[city_id].airports->GetEntry(j) ;
    node_id = data->id ;
    SendMessage(hwndCombo, CB_ADDSTRING, 0,
            (LPARAM) G_node_array[node_id].rics) ;
    }

  // Make the first entry in the combo box the default selection
  SendMessage(hwndCombo, CB_SETCURSEL, 0, 0) ;

} // end of handle_ArrivingCity_listbox

void handle_Case3_radiobutton(HWND hDlg)
{
HWND hwndList = GetDlgItem(hDlg, IDC_ArrivingCity);
HWND hwndCombo = GetDlgItem(hDlg, IDC_ArrivingAirport);
HWND hwndButton = GetDlgItem(hDlg, IDC_Execute);

  EnableWindow(hwndList, TRUE) ;
  EnableWindow(hwndCombo, TRUE) ;
  EnableWindow(hwndButton, TRUE) ;

} // end of handle_Case3_radiobutton

void handle_DepartingAirport_combobox(HWND hDlg, LPARAM lParam)
```

```
{
HWND hwndCombo = (HWND) lParam;
HWND hwndEdit = GetDlgItem(hDlg, IDC_Participants);
int index;
int hash_val, node_id;
char buffer[80];

  // Get the current selection
  index = SendMessage(hwndCombo, CB_GETCURSEL, 0, 0) ;

  // Get the rics (airport) of the current selection
  SendMessage(hwndCombo, CB_GETLBTEXT, index, (LPARAM) buffer) ;

  // Obtain the node (airport) index and display the number of participants
  // departing that node in the participants edit control.
  hash_val = G_airports->find_entry_in_lookup_table(buffer) ;
  node_id = G_airports->get_node_id(hash_val) ;
  sprintf(buffer, "%d", G_node_array[node_id].participants_temp) ;
  SetWindowText(hwndEdit, buffer) ;

} // end of handle_DepartingAirport_combobox

void handle_DepartingCity_listbox(HWND hDlg, LPARAM lParam)
{
HWND hwndList = (HWND) lParam;
HWND hwndEdit = GetDlgItem(hDlg, IDC_Participants);
HWND hwndCombo = GetDlgItem(hDlg, IDC_DepartingAirport);
int i;
char buffer[80];

  if (SendMessage(hwndCombo, CB_GETCOUNT, 0, 0) > 0)
    {
    save_number_of_participants(hwndEdit, hwndCombo) ;

    // Clear out the departing airport combo box
    SendMessage(hwndCombo, CB_RESETCONTENT, 0, 0) ;
    }

  // Clear out the number of participants edit control
  SendMessage(hwndEdit, EM_SETSEL, 0, -1) ;
  SendMessage(hwndEdit, WM_CLEAR, 0, 0) ;

  for (i = 0; i < G_number_cities; i++)
    {
    int j;
    int city_id;
int hash_val, node_id;
int combo_box_entry;
char *tokptr;
char city_name[12], short_name[10];
airport_record *data;

if (SendMessage(hwndList, LB_GETSEL, i, 0))
  {
  // City i has been selected

  if (!G_city_is_selected[i])
    {
    // Add airport(s) to combo list
    G_city_is_selected[i] = TRUE ;

    // Get the city and state from the list box’s selection
    SendMessage(hwndList, LB_GETTEXT, i, (LPARAM) buffer) ;

    // Extract the city name
    tokptr = strtok(buffer, ",") ;

    // Truncate the city name to first nine charaters
    memset(short_name, ‘\0’, 10) ;
    strncpy(short_name, tokptr, 9) ;

    // Go to the two-letter state abbreviation
    tokptr = strtok(NULL, " ") ;
    skip_leading_whitespace(&tokptr);

    // Append state to city string for hash table search
    sprintf(city_name, "%s%2s", short_name, tokptr) ;

    // Obtain the city index
    hash_val = G_cities->find_entry_in_lookup_table(city_name) ;
    city_id = G_cities->get_city_id(hash_val) ;

    combo_box_entry = 0;

    for (j = 1; j <= G_city_array[city_id].airports->GetNumberEntries(); j++)
      {
      data = G_city_array[city_id].airports->GetEntry(j) ;
      node_id = data->id ;
      SendMessage(hwndCombo, CB_ADDSTRING, 0,
              (LPARAM) G_node_array[node_id].rics) ;
      if (G_node_array[node_id].participants_temp)
        combo_box_entry = j - 1 ;
```

```
           }

         // Make combo_box_entry the default selection
         SendMessage(hwndCombo, CB_SETCURSEL, combo_box_entry, 0) ;

         // Get the rics of the default selection
         SendMessage(hwndCombo, CB_GETLBTEXT, combo_box_entry,
                (LPARAM) buffer) ;

         hash_val = G_airports->find_entry_in_lookup_table(buffer) ;
         node_id = G_airports->get_node_id(hash_val) ;
         sprintf(buffer, "%d", G_node_array[node_id].participants_temp) ;

         SetWindowText(hwndEdit, buffer) ;
         break ;
         }
       }
    else
      {
      // City i is not selected

      if (G_city_is_selected[i])
        G_city_is_selected[i] = FALSE ;
      }
    }

} // end of handle_DepartingCity_listbox

void handle_Execute_button(HWND hDlg)
{
HWND hwndButton = GetDlgItem(hDlg, IDC_Execute) ;
HWND hwndEditDur = GetDlgItem(hDlg, IDC_Duration);
HWND hwndEditPart = GetDlgItem(hDlg, IDC_Participants);
HWND hwndList = GetDlgItem(hDlg, IDC_DepartingCity);
HWND hwndRadio1 = GetDlgItem(hDlg, IDC_Case1);
HWND hwndRadio2 = GetDlgItem(hDlg, IDC_Case2);
HWND hwndRadio3 = GetDlgItem(hDlg, IDC_Case3);
HWND hwndComboDep = GetDlgItem(hDlg, IDC_DepartingAirport);
HWND hwndComboArr = GetDlgItem(hDlg, IDC_ArrivingAirport);
BOOL zero_departure_pts_specified = TRUE;
int i, j;
int hash_val, node_id;
char amount[10];
char buffer[80];

 // Get the conference duration from the duration edit control.
```

```
     GetWindowText(hwndEditDur, amount, GetWindowTextLength(hwndEditDur)+1) ;

     if ((G_conf_duration = atoi(amount)) < 1)
       {
       MessageBox (hDlg, "Conference duration is not greater than 0",
               "", MB_OK | MB_ICONWARNING ) ;
       return ;
       }

     save_number_of_participants(hwndEditPart, hwndComboDep) ;

//   if (SendMessage(hwndComboDep, CB_GETCOUNT, 0, 0) > 0)
//     {
     // Get the number of participants from the participants edit control.
//     GetWindowText(hwndEditPart, amount, GetWindowTextLength(hwndEditPart)+1)
;

     // Get the current selection from the departing airport combo box
//     i = SendMessage(hwndComboDep, CB_GETCURSEL, 0, 0) ;

     // Get the rics of the current selection
//     SendMessage(hwndComboDep, CB_GETLBTEXT, i, (LPARAM) buffer) ;

     // Obtain the node (airport) index and set the number of participants
     // departing that node.
//     hash_val = G_airports->find_entry_in_lookup_table(buffer) ;
//     node_id = G_airports->get_node_id(hash_val) ;
//     G_node_array[node_id].participants_temp = atoi(amount) ;
//     }

    for (i = 1; i <= G_number_nodes; i++)
      {
      G_node_array[i].participants_solve = 0 ;
      G_node_array[i].is_site = FALSE ;
      }

    for (i = 0; i < G_number_cities; i++)
      {
      int city_id;
      char city_name[12], short_name[10];
      char *tokptr;
      airport_record *data;

      if (SendMessage(hwndList, LB_GETSEL, i, 0))
        {
        // City i has been selected
      // Get the city and state from the list box's selection
      SendMessage(hwndList, LB_GETTEXT, i, (LPARAM) buffer) ;

      // Extract the city name
      tokptr = strtok(buffer, ",") ;

      // Truncate the city name to first nine charaters
      memset(short_name, '\0', 10) ;
      strncpy(short_name, tokptr, 9) ;

      // Go to the two-letter state abbreviation
      tokptr = strtok(NULL, " ") ;
      skip_leading_whitespace(&tokptr);

      // Append state to city string for hash table search
      sprintf(city_name, "%s%2s", short_name, tokptr) ;

      // Obtain the city index
      hash_val = G_cities->find_entry_in_lookup_table(city_name) ;
      city_id = G_cities->get_city_id(hash_val) ;

      for (j = 1; j <= G_city_array[city_id].airports->GetNumberEntries(); j++)
        {
        data = G_city_array[city_id].airports->GetEntry(j) ;
        node_id = data->id ;

        if (G_node_array[node_id].participants_temp)
          {
          // Airport has been selected
          zero_departure_pts_specified = FALSE ;
          G_node_array[node_id].participants_solve =
                          G_node_array[node_id].participants_temp ;
          }
        }
      }
    } // loop to the next entry in the departing city list box

  if (zero_departure_pts_specified)
    {
    MessageBox (hDlg, "No departure points have been specified or\nzero participants
from all departure points",
            "", MB_OK | MB_ICONWARNING ) ;
    return ;
    }

  if (SendMessage(hwndRadio1, BM_GETCHECK, 0, 0))      //Case 1
```

```
    {
    // Conference sites restricted to departing cities/airports
    for (i = 1; i <= G_number_nodes; i++)
      {
      if (G_node_array[i].participants_solve)
        {
        // Participants departing from airport i
        G_node_array[i].is_site = TRUE;
        }
      }
    }
  else if (SendMessage(hwndRadio2, BM_GETCHECK, 0, 0))  // Case 2
    {
    // All nodes considered as conference sites
    for (i = 1; i <= G_number_nodes; i++)
      G_node_array[i].is_site = TRUE;
    }
  else if (SendMessage(hwndRadio3, BM_GETCHECK, 0, 0))  // Case 3
    {
    // Single conference site specified

    if (SendMessage(hwndComboArr, CB_GETCOUNT, 0, 0) < 1)
      {
      MessageBox (hDlg, "You chose the third Solve option but\ndid not specify the
arriving airport",
              "", MB_OK | MB_ICONWARNING ) ;
      return ;
      }

    // Get the current selection from the arriving airport combo box
    j = SendMessage(hwndComboArr, CB_GETCURSEL, 0, 0) ;

    // Get the rics of the current selection
    SendMessage(hwndComboArr, CB_GETLBTEXT, j, (LPARAM) buffer) ;

    // Obtain the node (airport) index and set that node's conference site
    // flag.
    hash_val = G_airports->find_entry_in_lookup_table(buffer) ;
    node_id = G_airports->get_node_id(hash_val) ;
    G_node_array[node_id].is_site = TRUE;
    }

  SetCursor(LoadCursor(NULL, IDC_WAIT));
  ShowCursor(TRUE);

  EnableWindow(hwndButton, FALSE) ;
  determine_travel_costs();

  ShowCursor(FALSE);
  SetCursor(LoadCursor(NULL, IDC_ARROW));

  CreateDialog(G_hInstance, MAKEINTRESOURCE(IDD_SummaryDialog),
          hDlg, (DLGPROC) summary_dlg_proc);

  EnableMenuItem(GetMenu(GetParent(hDlg)), IDM_SaveAs, MF_ENABLED) ;
  InvalidateRect(GetParent(hDlg), NULL, TRUE) ;

} // end of handle_Execute_button

void save_number_of_participants(HWND hwndEdit, HWND hwndCombo)
{
int index;
int hash_val, node_id;
char amount[10];
char buffer[80];

  if (SendMessage(hwndCombo, CB_GETCOUNT, 0, 0) < 1) return ;

  // Get the number of participants from the participants edit control.
  // (One is added to GetWindowTextLength because for some reason the
  // function returns one less than the actual length of the text in the
  // edit control.)
  GetWindowText(hwndEdit, amount, GetWindowTextLength(hwndEdit)+1) ;

  // Get the current selection from the departing airport combo box
  index = SendMessage(hwndCombo, CB_GETCURSEL, 0, 0) ;

  // Get the rics of the current selection
  SendMessage(hwndCombo, CB_GETLBTEXT, index, (LPARAM) buffer) ;

  // Obtain the node (airport) index and set the number of participants
  // departing that node.
  hash_val = G_airports->find_entry_in_lookup_table(buffer) ;
  node_id = G_airports->get_node_id(hash_val) ;
  G_node_array[node_id].participants_temp = atoi(amount) ;

} // end of save_number_of_participants

/*-------------------------------------------------------
  winmain.cpp
 -------------------------------------------------------*/
```

```
#include <windows.h>
#define __COMMON_OFFSITE__
#include "offsite.h"
#include "offsresource.h"

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
            PSTR szCmdLine, int iCmdShow)
   {
   static char szAppName[] = "Off-Site" ;
   HWND       hwnd ;
   MSG        msg ;
   WNDCLASSEX  wndclass ;

   wndclass.cbSize        = sizeof (wndclass) ;
   wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
   wndclass.lpfnWndProc   = WndProc ;
   wndclass.cbClsExtra    = 0 ;
   wndclass.cbWndExtra    = 0 ;
   wndclass.hInstance     = hInstance ;
   wndclass.hIcon         = LoadIcon (hInstance, MAKEINTRESOURCE(ICO_Network))
;
   wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
   wndclass.hbrBackground = (HBRUSH) GetStockObject (WHITE_BRUSH) ;
   wndclass.lpszMenuName  = NULL ;
   wndclass.lpszClassName = szAppName ;
   wndclass.hIconSm       = LoadIcon (hInstance,
MAKEINTRESOURCE(ICO_Network)) ;

   RegisterClassEx (&wndclass) ;

   hwnd = CreateWindow (szAppName,         // window class name
                  szAppName,         // window caption
          WS_OVERLAPPEDWINDOW,      // window style
          0,                 // initial x position
          0,                 // initial y position
          GetSystemMetrics(SM_CXFULLSCREEN),        // initial x size
          GetSystemMetrics(SM_CYFULLSCREEN),        // initial y size
          NULL,                 // parent window handle
          LoadMenu(hInstance,
               MAKEINTRESOURCE(IDM_MainMenu)),     // window menu handle
          hInstance,              // program instance handle
                  NULL) ;                         // creation parameters

   ShowWindow (hwnd, iCmdShow) ;
   UpdateWindow (hwnd) ;
```

```
    while (GetMessage (&msg, NULL, 0, 0))
        {
        TranslateMessage (&msg) ;
        DispatchMessage (&msg) ;
        }
    return msg.wParam ;
    }

#include <windows.h>
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "offsite_hash.h"

city_table :: city_table()
{
int i;

  for(i = 0; i < CITY_TABLE_SIZE; i++)
     {
     city[i].city_id = NULL;
     memset(city[i].name, '\0', 12) ;
     }
}

int city_table :: hash_value(char *text)
{
int i;
int hash_val = 0;
int incrementer = 1;

  // Obtain the sum of characters in the supplied text
  for(i = 0; text[i] != '\0'; i++)
     hash_val += text[i] * incrementer++;

  // Return the hash value modulo CITY_TABLE_SIZE

  return ((hash_val * CITY_TABLE_DISPERSION_FACTOR) %
CITY_TABLE_SIZE);
}

int city_table :: find_entry_in_lookup_table(char *name)
{
int i, inc;
```

```
  i = this->hash_value(name);
  inc = 1;
  for(;;)
    {
    // If the entry for this index has no name assigned then this
    // name is not in the table
    if (city[i].name[0] == '\0') return (-1);

    if (strcmp(name, city[i].name) == 0) return(i);

    i += inc++;
    i %= CITY_TABLE_SIZE;
    if (inc > CITY_TABLE_SIZE)
      {
      return(-1);
      }
    }
}

int city_table :: insert_entry_in_lookup_table(char *name)
{
int i, inc;

  i = this->hash_value(name);
  inc = 1;
  for(;;)
    {

    // If the entry for this index has no symbol assigned then this
    // name is not in the table
    if (city[i].name[0] == '\0')
      {
      strncpy(city[i].name, name, 11);
      return(i);
      }

    if (strcmp(name, city[i].name) == 0) return(i);

    i += inc++;
    i %= CITY_TABLE_SIZE;
    if (inc > CITY_TABLE_SIZE)
      {
      char message[256] ;
      sprintf (message, "Hash table overflow on %s", name) ;
      MessageBox (NULL, message, "", MB_OK | MB_ICONWARNING ) ;
      exit(-1);
      }
    }
}

#ifndef __offsite_hash_h__
#define __offsite_hash_h__

// The maximum number of entries for the airport and city hash tables.
#define AIRPORT_TABLE_SIZE 2048
#define CITY_TABLE_SIZE    2048

// The dispersion constant used in calculating a hash value.
#define AIRPORT_TABLE_DISPERSION_FACTOR 42
#define CITY_TABLE_DISPERSION_FACTOR    42

class airport_table
{
private:
  typedef struct
    {
    char    airport_id[4];
    unsigned node_id;
    } airport_designator;

  airport_designator rics[AIRPORT_TABLE_SIZE];

public:
  airport_table();
  ~airport_table() {};

  int hash_value(char *text);
  int find_entry_in_lookup_table(char *name);
  int insert_entry_in_lookup_table(char *name, int node_id);
  int get_node_id(int hash_val) { return rics[hash_val].node_id; };
};

class city_table
{
private:
  typedef struct
    {
    char    name[12];
    unsigned city_id;
    } city_info;
  city_info city[CITY_TABLE_SIZE];

public:
  city_table();
  ~city_table() {};

  int hash_value(char *text);
  int find_entry_in_lookup_table(char *name);
  int insert_entry_in_lookup_table(char *name);
  int get_city_id(int hash_val) { return city[hash_val].city_id; };
  void set_city_id(int hash_val, int id) {city[hash_val].city_id = id; };
};

#endif

#include <windows.h>
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "offsite_hash.h"

airport_table :: airport_table()
{
int i, j;

  for(i = 0; i < AIRPORT_TABLE_SIZE; i++)
    {
    rics[i].node_id = NULL;
    memset(rics[i].airport_id, '\0', 4) ;
    }
}

int airport_table :: hash_value(char *text)
{
int i;
int hash_val = 0;
int incrementer = 1;

  // Obtain the sum of characters in the supplied text
  for(i = 0; text[i] != '\0'; i++)
    hash_val += text[i] * incrementer++;

  // Return the hash value modulo AIRPORT_TABLE_SIZE
  return ((hash_val * AIRPORT_TABLE_DISPERSION_FACTOR) %
AIRPORT_TABLE_SIZE);
}

int airport_table :: find_entry_in_lookup_table(char *name)
{
int i, inc;

  i = this->hash_value(name);
  inc = 1;
  for(;;)
    {
    // If the entry for this index has no name assigned then this
    // name is not in the table
    if (rics[i].airport_id[0] == '\0') return (-1);

    if (strcmp(name, rics[i].airport_id) == 0) return(i);

    i += inc++;
    i %= AIRPORT_TABLE_SIZE;
    if (inc > AIRPORT_TABLE_SIZE)
      {
      return(-1);
      }
    }
}

int airport_table :: insert_entry_in_lookup_table(char *name, int node_id)
{
int i, inc;

  i = this->hash_value(name);
  inc = 1;
  for(;;)
    {

    // If the entry for this index has no symbol assigned then this
    // name is not in the table
    if (rics[i].airport_id[0] == '\0')
      {
      strncpy(rics[i].airport_id, name, 3);
      rics[i].node_id = node_id;
      return(i);
      }

    if (strcmp(name, rics[i].airport_id) == 0) return(i);
    i += inc++;
    i %= AIRPORT_TABLE_SIZE;
    if (inc > AIRPORT_TABLE_SIZE)
      {
      char message[256] ;
      sprintf (message, "Hash table overflow on %s", name) ;
      MessageBox (NULL, message, "", MB_OK | MB_ICONWARNING ) ;
      exit(-1);
      }
    }
}


/*----------------------------------------------------------
  winproc.cpp
 ----------------------------------------------------------*/

#include <windows.h>
#include "offsite.h"
#include "offsresource.h"
#include "read.h"

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam,
LPARAM lParam)
{
static HWND hwndStatusBar ;
static OPENFILENAME ofn ;
static int cxClient ;  // width of client area
static int cyClient ;  // height of client area
HMENU hMenu = GetMenu(hwnd) ;
HDC   hdc ;
BOOL  show_names, show_map;

  switch (iMsg)
    {
    case WM_CREATE :
       {
       G_hInstance = ((LPCREATESTRUCT) lParam)->hInstance ;

       G_airports = new airport_table();
       G_cities = new city_table();
       G_number_cities = 0;
       G_city_is_selected = NULL;

       char arc_filename[] = "..\\data\\airport_arc.dat";
```

```
        char drv_cost_filename[] = "..\\data\\airport_drive_cost.dat";
        char flt_cost_filename[] = "..\\data\\airport_fly_cost.dat";
        char map_filename[] = "..\\maps\\usa.map";
        char node_filename[] = "..\\data\\airport_node.dat";
        char per_diem_filename[] = "..\\data\\per_diem.dat";
        char route_filename[] = "..\\data\\airport_route.txt";

        read_usa_map(map_filename);
        read_arc_file(arc_filename);
        read_node_file(node_filename);
        read_per_diem_file(per_diem_filename);
        read_flight_cost(flt_cost_filename);
        read_drive_cost(drv_cost_filename);
        read_route_file(route_filename);

        // Create status bar
        hwndStatusBar = init_status_bar (hwnd) ;

        init_openfilename_struc (hwnd, &ofn) ;

        CheckMenuItem(hMenu, IDM_AirportNames , MF_BYCOMMAND |
MF_CHECKED) ;
        CheckMenuItem(hMenu, IDM_Map , MF_BYCOMMAND | MF_CHECKED) ;

        return 0 ;
        }

    case WM_COMMAND :
        switch (LOWORD (wParam))
          {
          case IDM_SaveAs :
              {
              char buffer[256];
              char szFile[] = "noname.txt" ;
              FILE *read_fp, *write_fp;

              ofn.hwndOwner = hwnd ;
              ofn.lpstrFile = szFile ;
              ofn.Flags = OFN_HIDEREADONLY | OFN_OVERWRITEPROMPT ;

              if (GetSaveFileName(&ofn))
                {
                if ((read_fp = fopen(G_cost_filename, "r")) == NULL ||
                   (write_fp = fopen(ofn.lpstrFile, "w")) == NULL)
                  {
                  sprintf (buffer, "Can't save %s", ofn.lpstrFile) ;
        MessageBox (hwnd, buffer, "", MB_OK | MB_ICONWARNING ) ;
        return 0;
        }

      while(fgets(buffer, sizeof(buffer), read_fp) != NULL)
        {
        fprintf(write_fp, "%s", buffer);
        }

      fclose (read_fp) ;
      fclose (write_fp) ;
      }

    }
    return 0;

case IDM_Exit :
    SendMessage(hwnd, WM_CLOSE, 0, 0L);
    return 0 ;

case IDM_Run :
    EnableMenuItem(hMenu, IDM_Run, MF_GRAYED) ;
    CreateDialog(G_hInstance, MAKEINTRESOURCE(IDD_SolveDialog),
              hwnd, (DLGPROC) solve_dlg_proc) ;
    return 0 ;

case IDM_AirportNames :

    // Retrieve the state of the Show Airport Names item
    show_names = GetMenuState(hMenu, IDM_AirportNames,
                        MF_BYCOMMAND) & MF_CHECKED ;

    // Toggle the state of the item
    CheckMenuItem(hMenu, IDM_AirportNames , MF_BYCOMMAND |
              (show_names ? MF_UNCHECKED : MF_CHECKED)) ;

    InvalidateRect(hwnd, NULL, TRUE) ;
    return 0 ;

case IDM_Map :
    // Retrieve the state of the Show Map item
    show_map = GetMenuState(hMenu, IDM_Map,
                        MF_BYCOMMAND) & MF_CHECKED ;

    // Toggle the state of the item
    CheckMenuItem(hMenu, IDM_Map , MF_BYCOMMAND |
                    (show_map ? MF_UNCHECKED : MF_CHECKED)) ;

              InvalidateRect(hwnd, NULL, TRUE) ;
              return 0 ;

          }
        break;

    case WM_LBUTTONDOWN :
        return 0 ;

    case WM_MOUSEMOVE :
        return 0 ;

    case WM_LBUTTONUP :
        return 0 ;

    case WM_PAINT :
        {
        PAINTSTRUCT ps ;
        int i, site_index ;
        int extent ;
        RECT rect ;
        HPEN hPen[NUMBER_OF_PENS] ;

        hPen[PEN_BLACK] = CreatePen(PS_SOLID, 1, RGB(0, 0, 0)) ;
        hPen[PEN_RED] = CreatePen(PS_SOLID, 1, RGB(255, 0, 0)) ;
        hPen[PEN_GREEN] = CreatePen(PS_SOLID, 1, RGB(0, 255, 0)) ;
        hPen[PEN_BLUE] = CreatePen(PS_SOLID, 1, RGB(0, 0, 255)) ;

        extent = max(G_map_width, G_map_height) / 2.0 ;

              hdc = BeginPaint (hwnd, &ps) ;

        SetMapMode(hdc, MM_ISOTROPIC) ;
        SetWindowExtEx(hdc, extent, extent, NULL) ;
        SetViewportExtEx(hdc, -cxClient / 2, -cyClient / 2, NULL) ;
        SetViewportOrgEx(hdc, cxClient / 2, cyClient / 2, NULL) ;

        // Retrieve the state of the Show Map item
        show_map = GetMenuState(hMenu, IDM_Map,
                    MF_BYCOMMAND) & MF_CHECKED ;

        G_map_entities->Draw(hdc, show_map, hPen);

#ifndef CHECK_MAP_ALIGNMENT
```

```
        if ((site_index = G_best_sites[1].site) < 1)
          {
          EndPaint (hwnd, &ps) ;

          for (i = 0; i < NUMBER_OF_PENS; i++)
            DeleteObject(hPen[i]) ;

          return 0 ;
          }
#endif
        // Retrieve the state of the Show Airport Names item
        show_names = GetMenuState(hMenu, IDM_AirportNames,
                          MF_BYCOMMAND) & MF_CHECKED ;

#ifndef CHECK_MAP_ALIGNMENT
        for (i = 1; i <= G_number_nodes; i++)
          {
          int index;

          if ((i != site_index) && G_node_array[i].participants_solve)
            {
            if (i < site_index)
              {
              index = site_index - i;
              G_node_array[i].cost[index].path->Draw(i, site_index,
                                     hdc, show_names,
                                     hPen) ;
              }
            else
              {
              index = i - site_index;
              G_node_array[site_index].cost[index].path->
                                  Draw(i, site_index,
                                     hdc, show_names,
                                     hPen) ;
              }
            }
          }

        for (i = 1; i <= G_number_nodes; i++)
          {
          if ((i != site_index) && G_node_array[i].participants_solve)
            {
            SelectObject(hdc, hPen[PEN_BLACK]) ;
            rect.left = (long) G_node_array[i].x - 20 ;
            rect.top = (long) G_node_array[i].y + 20 ;
            rect.right = (long) G_node_array[i].x + 20 ;
            rect.bottom = (long) G_node_array[i].y - 20 ;

            Ellipse (hdc, rect.left, rect.top, rect.right, rect.bottom) ;

            if (show_names)
              {
              TextOut(hdc, rect.left, rect.top, G_node_array[i].rics,
                    strlen(G_node_array[i].rics)) ;
              }
            }
          }

        rect.left = (long) G_node_array[site_index].x - 20 ;
        rect.top = (long) G_node_array[site_index].y + 20 ;
        rect.right = (long) G_node_array[site_index].x + 20 ;
        rect.bottom = (long) G_node_array[site_index].y - 20 ;

        SelectObject(hdc, hPen[PEN_RED]) ;
        Ellipse (hdc, rect.left, rect.top, rect.right, rect.bottom) ;

        if (show_names)
          {
          TextOut(hdc, rect.left, rect.top, G_node_array[site_index].rics,
                strlen(G_node_array[site_index].rics)) ;
          }
#else
        for (i = 1; i <= G_number_nodes; i++)
          {
          SelectObject(hdc, hPen[PEN_BLACK]) ;
          rect.left = (long) G_node_array[i].x - 20 ;
          rect.top = (long) G_node_array[i].y + 20 ;
          rect.right = (long) G_node_array[i].x + 20 ;
          rect.bottom = (long) G_node_array[i].y - 20 ;

          Ellipse (hdc, rect.left, rect.top, rect.right, rect.bottom) ;

          if (show_names)
            {
            TextOut(hdc, rect.left, rect.top, G_node_array[i].rics,
                  strlen(G_node_array[i].rics)) ;
            }
          }
#endif
            EndPaint (hwnd, &ps) ;
```

```
    for (i = 0; i < NUMBER_OF_PENS; i++)
      DeleteObject(hPen[i]) ;

    return 0 ;
    }

case WM_SIZE :
    {
    int cyStatus ;
    RECT rWindow ;

    cxClient = LOWORD (lParam) ;
    cyClient = HIWORD (lParam) ;

    // Adjust status bar size
    GetWindowRect (hwndStatusBar, &rWindow) ;
    cyStatus = rWindow.bottom - rWindow.top ;
    MoveWindow (hwndStatusBar, 0, cyClient - cyStatus,
             cxClient, cyStatus, TRUE) ;
    return 0 ;
    }

case WM_DESTROY :
    if (G_city_is_selected != NULL)
      delete [] G_city_is_selected ;

    delete [] G_arc_array ;

    for (int i = 1; i < G_number_nodes; i++)
      {
      int number_of_destins = G_number_nodes - i ;

      for (int j = 1; j <= number_of_destins; j++)
        {
        if (G_node_array[i].cost[j].path != NULL)
          delete G_node_array[i].cost[j].path ;
        }

      delete [] G_node_array[i].cost ;

      if (G_city_array[i].airports != NULL)
        delete G_city_array[i].airports ;
      }

    delete [] G_node_array ;
    delete [] G_best_sites ;
```

```
          delete G_cities ;
          delete G_airports ;
          delete G_map_entities ;

          PostQuitMessage (0) ;
          return 0 ;
       }

    return DefWindowProc (hwnd, iMsg, wParam, lParam) ;
  }

  void init_openfilename_struc (HWND hwnd, OPENFILENAME *ofn)
  {
  static char szFilter[]= "Text Files (*.TXT)\0*.txt\0" \
                          "All Files (*.*)\0*.*\0\0" ;
  static char szInitialDir[] = "..\\out" ;

    ofn->lStructSize = sizeof (OPENFILENAME) ;
    ofn->hwndOwner = hwnd ;
    ofn->hInstance = NULL ;
    ofn->lpstrFilter = szFilter ;
    ofn->lpstrCustomFilter = NULL ;
    ofn->nMaxCustFilter = 0 ;
    ofn->nFilterIndex = 0 ;
    ofn->lpstrFile = NULL ;
    ofn->nMaxFile = _MAX_PATH ;
    ofn->lpstrFileTitle = NULL ;
    ofn->nMaxFileTitle = _MAX_FNAME + _MAX_EXT ;
    ofn->lpstrInitialDir = szInitialDir ;
    ofn->lpstrTitle = NULL ;
    ofn->Flags = 0 ;
    ofn->nFileOffset = 0 ;
    ofn->nFileExtension = 0 ;
    ofn->lpstrDefExt = "txt" ;
    ofn->lCustData = 0L ;
    ofn->lpfnHook = NULL ;
    ofn->lpTemplateName = NULL ;
  }

#ifndef __offsite_h__
#define __offsite_h__

#ifndef __COMMON_OFFSITE__
#define __COMMON_OFFSITE__ extern
#endif
```

```
#include <windows.h>
#include <stdio.h>
#include "Tlist.h"
#include "offsite_hash.h"
#include "map_entity.h"

#ifdef FALSE
#undef FALSE
#endif

#define FALSE 0

#ifdef TRUE
#undef TRUE
#endif

#define TRUE 1

#define NUMBER_OF_PENS 4
#define PEN_BLACK      0
#define PEN_RED        1
#define PEN_GREEN      2
#define PEN_BLUE       3

typedef struct
  {
  unsigned start_node;
  unsigned end_node;
  int      cost;
  } arc;

typedef struct
  {
  unsigned node1;
  unsigned node2;
  } transit_record;

typedef struct
  {
  unsigned      amount;        // cost to fly or drive
  unsigned char  is_drive_cost; // cost represents driving cost (TRUE/FALSE)
  TPathList<transit_record>
                *path;         // list of nodes to visit while transiting
  } travel_cost;

typedef struct
```

```
  {
  char    rics[4];      // three-letter airport designator
  char    city[16];     // city name
  int     state;
  float   x;            // in kilometers
  float   y;            // in kilometers
  unsigned participants_temp;  // # of participants extracted from dialog box
  unsigned participants_solve; // # of participants actually travelling
  unsigned meals;        // meal per diem
  unsigned lodging;      // lodging per diem
  unsigned is_site;     // designated conference site (TRUE/FALSE)
  travel_cost *cost;    // travel cost to other nodes
  } node;

typedef struct
  {
  unsigned site;
  int     cost;
  } potential_site;

typedef struct
  {
  unsigned id;          // index to node
  } airport_record;

typedef struct
  {
  char city[16];
  int  state;
  TList<airport_record> *airports; // list of airports
  } city_and_airport;

__COMMON_OFFSITE__ char *state_code[] =
  {
  "AK", "AL", "AR", "AZ", "CA", "CO", "CT", "DC", "DE", "FL", "GA", "HI",
  "IA", "ID", "IL", "IN", "KS", "KY", "LA", "MA", "MD", "ME", "MI", "MN",
  "MO", "MS", "MT", "NC", "ND", "NE", "NH", "NJ", "NM", "NV", "NY", "OH",
  "OK", "OR", "PA", "PR", "RI", "SC", "SD", "TN", "TX", "UT", "VA", "VT",
  "WA", "WI", "WV", "WY"
  };

__COMMON_OFFSITE__ char G_cost_filename[] = "..\\out\\travel_cost.txt";

__COMMON_OFFSITE__ unsigned G_number_arcs;
__COMMON_OFFSITE__ unsigned G_number_nodes;
__COMMON_OFFSITE__ unsigned G_number_cities;
__COMMON_OFFSITE__ arc *G_arc_array;
__COMMON_OFFSITE__ node *G_node_array;

__COMMON_OFFSITE__ unsigned G_conf_duration;

__COMMON_OFFSITE__ unsigned G_number_sites;
__COMMON_OFFSITE__ potential_site *G_best_sites;

__COMMON_OFFSITE__ airport_table *G_airports;
__COMMON_OFFSITE__ city_table    *G_cities;

__COMMON_OFFSITE__ city_and_airport *G_city_array;

__COMMON_OFFSITE__ BOOL *G_city_is_selected;

__COMMON_OFFSITE__ float G_map_width;  // in kilometers
__COMMON_OFFSITE__ float G_map_height; // in kilometers

__COMMON_OFFSITE__ TMapEntityList<map_entity> *G_map_entities;

__COMMON_OFFSITE__ HINSTANCE G_hInstance;

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);
HWND init_status_bar (HWND hwndParent);

void init_openfilename_struc (HWND hwnd, OPENFILENAME *ofn);
void save_as_dlg(HWND hwnd, OPENFILENAME *ofn);

BOOL CALLBACK solve_dlg_proc (HWND, UINT, WPARAM, LPARAM);
void disable_Arriving_controls(HWND);
void handle_ArrivingCity_listbox(HWND, LPARAM);
void handle_Case3_radiobutton(HWND);
void handle_DepartingCity_listbox(HWND, LPARAM);
void handle_DepartingAirport_combobox(HWND, LPARAM);
void handle_Execute_button(HWND);
void save_number_of_participants(HWND, HWND);

BOOL CALLBACK summary_dlg_proc (HWND, UINT, WPARAM, LPARAM);

void determine_travel_costs(void);
int tdy_cost_to_site(unsigned site, unsigned origin, FILE *cost_fp);

#endif
```

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE
May 1998

3. REPORT TYPE AND DATES COVERED
Final

4. TITLE AND SUBTITLE
Defense Contract Management Command Site Selection Model

5. FUNDING NUMBERS

6. AUTHOR(S)
Major Randy Zimmerman
Captain Jeffery L. Huisingh

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

DLA Office of Operations Research and
Resource Analysis (DORRA)
c/o Defense Supply Center Richmond
8000 Jefferson Davis Highway
Richmond, VA 23297-5082

8. PERFORMING ORGANIZATION REPORT NUMBER

DLA-98-PB80140

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Defense Contract Management Command
8726 John J. Kingman Road
Ft. Belvoir, VA 22060-6221

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Public Release; Unlimited Distribution

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

A recurrent reason for DCMC employee travel is to attend professional conferences or development training. The location of the conference or training site is often an arbitrary decision with little regard for travel cost. We developed a methodology to evaluate relevant travel costs and return the least expensive conference site given over 260 cities in the US with Government contracted airfares. This schema involves a new use of Dykstra's algorithm modified for non-capacitated shortest path network optimization as well as the computation of inter-airport distances using latitude and longitude data. This paper introduces a program called OffSite which optimizes the selection of collective training events or conferences where people need to come together from geographically dispersed locations. OffSite requires meeting or conference planners to input a list of origin cities for conference participants. It then determines the least cost location for hosting the conference. Additionally, OffSite is flexible enough to allow planners to choose a preferred destination. This system can also restrict the search for meeting venues to origin locations should an organizer wish to have one of the meeting's participants "host" the conference.

14. SUBJECT TERMS
Travel, Optimization

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
Unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

19. SECURITY CLASSIFICATION OF ABSTRACT
Unclassified

20. LIMITATION OF ABSTRACT